学習 BDSC· a-C

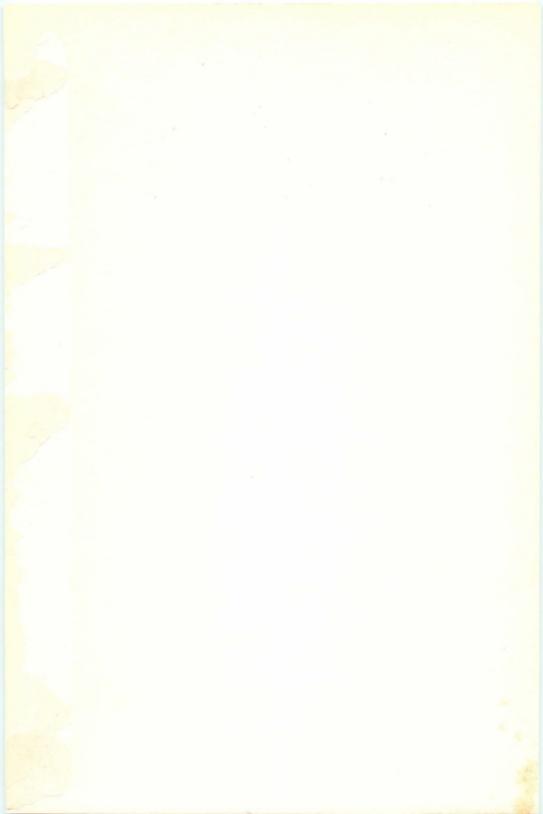
御手洗 毅・理英 著

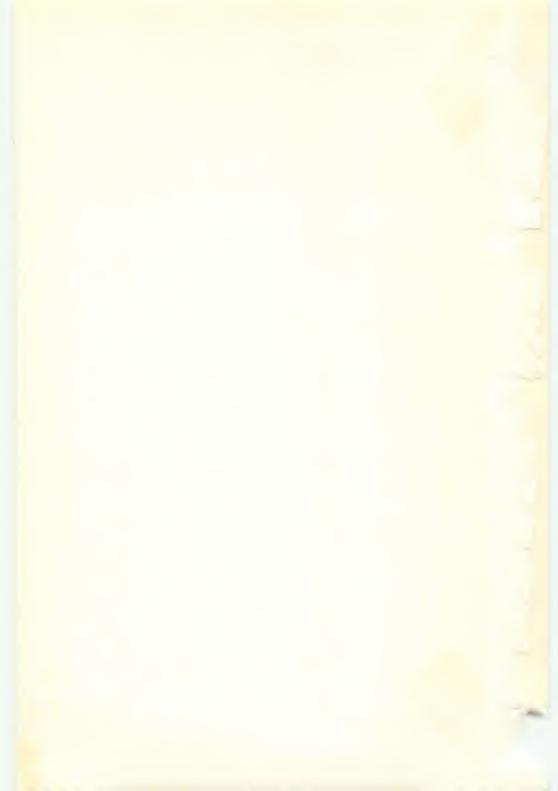
工学図書株式会社

- CP/Mは Digital Research Inc. の登録 商標です。
- BDS Cは BD SOFTWARE, Inc. の登録商標です。
 - 日本総代理店は(株)ライフボートです.
- α-Cは(株)ライフボートの商標です。

学習 BDSC·α-C

御手洗 毅•理英 著











学習 BDSC·α-C

御手洗 毅·理英 著

工学図書株式会社

o-wede

部手充一報·理英·音

其全先耕書図業工

まえがき

C言語は現在、パソコンの機種を越えた共通の言語として使われるようになり、学習する方も非常に多くなりました。中でも、BDS C、 α -C は 8 ビットパソコン上で動作する本格的な C コンパイラとして有名で、安価で操作が簡単なことから、初心者も気軽に学習をはじめる事ができます。

ただ、残念なことに、C言語はシステム記述言語と呼ばれるように、OSなど高度なプログラム開発を目的とした言語であるため、学習は簡単ではありません。さらに、BDS C は8 ビットのC言語として最もポピュラーなものであるにもかかわらず、標準的なC言語から一部機能の削除されたサブセット版であり、標準C の参考書では学習しにくい場合があることも事実です。

そのため、本書ではC言語そのものの解説は最小限に留め、BDSC、 α -Cを学習する上で判りにくいポイント、重要なポイントが把握できるようにサンプルプログラムを中心に解説しました。初心者の方には最適の教科書になると思われます。

さらに、応用編として BDS C、 α -C における日本語処理、オーバーレイ技法などを取り上げ、初心者だけでなく、本格的に BDS C を使っている方も十分役立つ内容となっています。

是非,本書を活用されることを期待しています.

なお,本書では次のバージョンを用いています。

BDS C バージョン1.50a

α-C バージョン1.51

昭和62年1月

- ■CP/Mは Digital Research Inc. の登録商標です.
- ■BDS CはBD SOFTWARE, Inc. の登録商標です。 日本総代理店は㈱ライフボートです。
- α-Cは(株)ライフボートの商標です。

目 次

第1章 BDS C入門····································					
1 - 1	BDS Cのセットアップ・・・・・・2				
1 - 2	初心者のためのサンプルプログラム9				
第 2 章 Cのプログラムスタイル					
2 - 1	ポインタの使い方 18				
2 - 2	コマンドラインの受け取り方 27				
2 - 3	ファイルの入出力				
2 - 4	文字列の処理・・・・・・・・・・・・・・・・・・・・・・・43				
2 - 5	構造体の使い方 51				
第3章 BDS C特有の問題点 57					
3 - 1	コンパイルエラー・・・・・・58				
3 - 2	BDS Cで削除されている機能 61				
3 - 3	プリプロセッサの機能 68				
3 - 4	浮動小数点演算 · · · · · · 77				
第4章 CI	P/M サンプルプログラム95				
4 - 1	拡張サブミットコマンド@.com ····· 96				
4 - 2	拡張サブミット補助コマンド /. com106				
第5章 日	本語処理				

	5 - 1	BDS C, α-Cの変更・・・・・・120		
	5 - 2	プリプロセッサを用いる方法125		
	5 - 3	日本語処理パッケージ133		
	5 - 4	日本語処理パッケージの使い方150		
第6	章 オ·	ーバーレイとチェイン153		
	6 - 1	プログラムのチェイン154		
	6 - 2	swapin 関数を用いたオーバーレイ159		
	6 - 3	オーバレイ関数の呼び出し方法164		
あとがき175				
コンパイルエラーメッセージー覧表177				
参考	文献⋯			
索	引	189		

第章

BDS C入門

BASICでは、パソコンの電源を入れてBASICを起動させ、そのあとプロ グラムが完成するまでフロッピーディスクを抜いたままで作業を行なう ことができます。これは、BASICインタプリンタ本体がパソコンのメモリ 上に常駐しており、その上で、プログラムの作成、変更、実行といった 一連のプログラム開発作業がすべて行なえるようになっているからです。 BDS CはCP/Mのもとで動作しますが、BDS Cに限らず、すべてのCP/ M用のソフトウェアはディスクが基本になっており、フロッピーディス クがなければ一切の作業を行なうことができません。エディタ、言語(コ ンパイラ) などは異なるプログラムとして独立してディスク内に記録さ れ、必要に応じて実行されるわけです。これは大変不便なようですが、 メモリに常駐している部分が小さいため、大きなマシン語プログラムを 動作させることができるという特徴を持っています。ちょっとしたプロ グラムを作るだけなら、BASICで十分という場合もありますが、本格的な プログラムを動作させたい、作成したいという場合には、CP/Mを用いる ことは常識であり、中でもBDS Cは本格的な8ビット用のソフトウェアを 作るのに向いています。

1-1 BDS Cのセットアップ

(1) BDS C, α-C 利用の手順

BDS C、α-Cを用いたプログラム作成の概要について説明します.

まず、C言語のソースファイルはテキストエディタを用いて作成します。 CP/Mを購入した際にED.COMというエディタが付属していますのでそれを 用いることもできますが、EDは1行ごとに表示させながら行単位で書いてい くラインエディターですので、一般には画面にプログラムを表示させ、カー ソルにより1文字ずつ編集するスクリーンエディタを利用する方が便利です。

CP/M-80用には米マイクロプロ社のワードマスターやワードスターが有名ですが、C言語のソースプログラムをディスクファイルとして作成できるものであれば何でも構いません。しかし、エディタが使えなければBDS Cは使うことができませんので、その操作に慣れることがC言語に慣れる第一歩と言っても過言ではないでしょう。

次に、作成したソースファイルを BDS C あるいは α-C コンパイラによってコンパイルします。コンパイラ本体は両者とも同じファイル名で、

cc.com

という2つから成り、これらがオーバーレイ(チェイン)してコンパイル作業を行ないます。ここで重要なのは、これらのプログラムで処理された出力はそのままCP/M上で実行できるわけではなく、一旦リロケータブルなバイナリ形式のファイルとしてディスク上に作成されることです。これは複数のソースファイルから1つの実行プログラムを作成したり、すでにBDSCに備わっている別のプログラムモジュール(標準関数など)と組み合せる際に

必要だからです。

実行ファイルを作成するためには、リンカーを用います。リンカーは

clink.com

という専用プログラムが用意されています。これを用いると、実行形式の COMファイルが得られ、作成したプログラムを実際に動作させてみることが できるわけです。cc, cc2, clink などの基本ソフトウェアはまず、マスター ディスクからピックアップし、作業用ディスクにコピーしておくことが必要 ですが、使用ディスクのタイプによって要領がかなり異なってきます 1MB 以上のディスクを使う場合には1基でも余裕がありますので、滅多に使わな いプログラムもコピーしておくことが可能ですが、320KB 程度のドライブの 場合にはどうしても最低限必要なファイルだけをまとめておき、それ以外の ファイルが必要になった時にはディスクを抜き差しして使うことになります 320KBドライブの場合、1基でも利用は可能ですが、大きなプログラムを作 るとすぐディスクが一杯になってしまいますので、実用的には2基以上必要 です

< 最低限必要なファイル (BDS C, α-C共通)>

cc.com

cc2.com

clink.com

deff.crl

deff2.crl

C.CCC

bdscio.h

〈必要に応じて利用するファイル (BDS C, α -C共通)〉

clib.com (Cライブラリアン)

dio.h	(ダイレクトコンソール IO パッケージ)
dio.crl	(ダイレクトコンソール IO パッケージ)

〈必要に応じて利用するファイル (BDS Cのみ)〉

float.crl	(浮動小数点演算パッケージ)
long.crl	(倍精度整数演算パッケージ)
casm.com	(アセンブラパッケージ)
casm.sub	(アセンブラパッケージ)
cdb.com	(シンボリックデバッガー)
cdb2.ovl	(シンボリックデバッガー)
12.com	(cdb 用リンカー)

【注意】 dio, float, long, cdb, l2などはマスターディスクにソースリストの形で格納されていますので、マニュアルに従って、あらかじめ処理する必要があります。また、casm を用いる際には CP/M の標準ユーティリティ ASM、DDTが必要になります。

〈最低限必要な CP/Mユーティリティ〉

PIP.COM

STAT. COM

SUBMIT. COM

XSUB.COM

ED. COM (あるいは他のエディタ)

その他,ディスクフォーマット・ディスクバックアップ用のユーティリティなど。

(2) ソースファイルの作成

必要なファイルを格納した作業用のディスクを作ると、プログラムの作成 を始めることができます。

まず、最初にC言語のソースファイルを作成します。ここでは最も簡単な

プログラムとして、画面に文字を出力するプログラムを作成してみます。

リスト1-1-1 hello.c

```
#include <bdscio.h>
main()
{
        printf ("hello, world\f");
}
```

この内容のファイルを CP/M付属のエディター, ED を用いて作ってみましょう. 下線部が実際に手でタイプした部分です.

操作例 1-1-2 hello.cの入力

A>ed hello.c

A>

EDには多くのコマンドがあり、ファイルの作成、修正、編集などが行なえるようになっていますが、詳細は省略します。

さて、プログラムはたった5行ですが、これはC言語の基本的なマニュアルである「プログラミング言語C」*という本の最初にでてくるサンプルプロ

^{*} 巻末の参考文献参照.

グラムで、おそらく現在C言語を使っている多くの方が1度は実際にタイプ してみた例の1つだと思います。

一目でわかるように、Cはその多くの部分を小文字で記述します。見た目に美しく、かつ {,} などの記号を用いることでタイプ量が少なくなっているのもその特徴の1つになっています。また、強制ではありませんが、対応関係をはっきりさせるために、インデント(字下げ)を行ないます。

さて、実際のプログラムですが、

#include <bdscio.h>

という第1行はBDS Cのプログラムを書く上での決まり文句です。初心者のうちは、第1行は必ずこのように書くと考えて構わないでしょう。実際には、ディスクから

bdscio.h

というファイルを読み込み、この行と入れ換える動作を行ないます。 次の行から実際のプログラムがスタートします。

main()

main はこの関数の名前です。関数というのは、C言語におけるプログラムの単位で、1つのサブルーチンと考えれば良く、()の中には通常引数を列挙します。たまたまここでは引数はありませんが、カッコを省略することはできません。そして実際のプログラム本体は次のカーリーブレイス記号 {から、最後の} までとなります。{ | はプログラムの中にも登場しますので、それらの対応関係が正しくなるように関数単位で | と | の数を合わせる必要があります。

なお、関数名はキーワード (ifや for など)・変数名とダブらなければ自由 に決めることができるのですが、mainという関数名だけは、固定 (変更は可能)で、プログラムそのものがスタートした時、必ずこのmain という関数 から実行が開始される決まりになっています。いくつも他の関数が定義されていて、main が最後の方にあったとしても実行は必ず main から始まるわけです。言い方を換えると、必ず関数 main は必要だということです。

printf ("hello,world\forman");

この1行が実際に動作するプログラムの本体です。"hello, world" という 文字列を画面に表示するわけですが、ここで用いている関数 printf は BASIC の PRINT 文などと異なり、 C言語の機能としてあらかじめ用意されている わけではなく、あくまで1つの関数として扱われ、ユーザーが作成した関数 との区別はありません。つまり、単に BDS C の作成者がユーザーの便利の ために作っておいたサブルーチンの1つなのです。

BASICではPRINT文などのあらかじめ備わっている機能と、ユーザーが 作成したサブルーチンは完全に異なるものであり、新たにステートメント(文) を追加・拡張したりすることができません。

さて、このように関数を利用する場合には、まず関数名を書き、()の中にその関数への引数を記述します。この場合は""で囲んだ文字列が引数ということになっているわけです。なお、文字列の最後の ¥n は改行の指定です。文字として書くことができない制御コードなどは、¥マークと文字を組み合せて表現することになっています。BASICで、

PRINT "hello" \(\xeta, \)
PRINT "hello";

の違いはお判りですね。BASICと逆に、改行する場合にこのような付加記号が必要になってくるのです。

なお, 文の終りには, 必ずセミコロン (;) を付けます.

それでは、実際のコンパイル作業を行なってみましょう。このプログラムは

hello.c

というファイル名で、ドライブAに作成しましたから、操作例1-1-3のようにコンパイルし、実行してみてください。

操作例 1-1-3 hello.cのコンパイル

A>cc hello hello.cのコンパイル、hello.crlというファイルが作成される。 BD Software C Compiler v1.50a (part I) 35K elbowroom BD Software C Compiler v1.50 (part II) 32K to spare

A>clink_hello hello.crlからhello.com という実行ファイルを作成する.
BD Software C Linker v1.50
Linkage complete
43K left over

A><u>hello</u> helloの実行. hello, world

A>

正しく実行できたでしょうか。これらのコンパイル、リンクの流れを図1-1-4に図示しておきます。

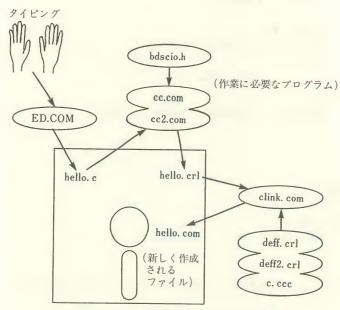


図1-1-4 BDS Cのコンパイル作業

1-2 初心者のためのサンプルプログラム

すべてのコンピュータ言語に共通して言えることですが、上達への早道は、 簡単な短いプログラムを数多く作ってみることです。本項では、学習に適し たサンプルプログラムを解説とともに示しますので、実際にタイプし、実行 してみてください。

(1) 1から10までを加算して表示するプログラム

リスト 1-2-1 goukei. cと実行例

#include <bdscio.h>

main()

{

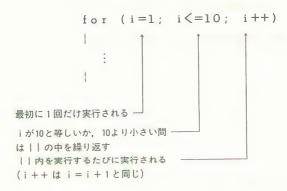
〈実行例〉

A>GOUKEI

GOUKEI = 55

A>

C言語の for 文は BASIC とは異なりますので、注意が必要です。



条件は制御変数 i と終値との比較だけでなく, i と全く関係のない比較文を書くこともできます.また,単純増加,減少だけでなく, i ++のかわりに i=i*2 などとすることもできますので,はるかに用途が広くなっています.また,条件のチェックはループの実行前に行なわれます.従って,最初から条件が成立しなければループ内部は一度も実行されません.

(2) 100までの素数を表示するプログラム

```
リスト1-2-2 sosuu.cと実行例
#include
                 <bdscio.h>
main()
         int
                  i,s;
         char
                  f:
         for (s=2; s<=100; s++)
                  f=0:
                  for (i=2; i<=s-1; i++)
                           if (s\%i = 0)
                                    f=1:
                                    break;
                          }
                  }
                 if (f==0)
                                   printf ("%d\t",s);
         }
}
〈実行例〉
A>SOSUU
2
      3
             5
                           11
                                  13
                                         17
                                                19
                                                       23
                                                              29
31
       37
             41
                    43
                           47
                                  53
                                         59
                                                61
                                                       67
                                                              71
73
      79
             83
                    89
                           97
A>
```

%は剰余演算子で、s%iでsをiで割った余りを求めます。

リスト1-2-2はある数に対し、2からその数より1小さい数までしらみつぶしに割り切れるかどうかを調べ、割り切れなかった場合を素数とします。この例では、割り切れた場合にフラグfを1にし、それ以降チェックを省略し

A>

ます。このようなループの中途強制中断には break を用います。

ループの外でフラグ f を見て、0 であれば割り切れる数がなかったのですから、素数として画面に表示します。

Yt はタブコードで、TAB キーを押したのと同様、数字と数字の間にスペースをあけます。Yn などと同じ文字列中の特殊文字です。

(3) 任意の2つの数字を手で入力し、その加算結果を表示するプログラム

リスト1-2-3 nyuryoku.cと実行例

scanf はBASICのINPUT文に相当します。しかし、そう瀬繁に用いられる関数ではありません。ここで、引数を指定する際に変数名の頭に&記号を付けていることに注目してください。 C言語では、関数の戻り値としてただ1つの値しか返すことができません。 2つ以上の値を返さねばならない場合、&記号

をつけて変数の物理的なアドレスを渡し、そのアドレス位置に書き込ませることによって、目的を達します。C言語ではアセンブラと同様、このようなアドレスをしばしば用います。他の高級言語には余り例がなく、C言語の特徴でもあります。2つの変数の値を入れ換えたりする場合にも引数を変数のアドレスにしなければ、実現できません。

(4) 手で入力した文字をそのまま画面に表示するプログラム

```
リスト 1-2-4 typeout.cと実行例
```

〈実行例〉

A> typeout

AABBCCDDEEFFGGHHIIJJKKLLMMNN

 2

A>

このプログラムはキーボードからの1文字入力関数 getcher と出力 関数 putchar の使用例ですが、getchar でその値を格納している変数 c が int (整数) 型であることに注意してください。ファイル終了コードと同じ 0x1a (コントロールZ)をキーボードから入力すると、入力の終了と判定し、getchar はー1を返しますが char(文字)型の変数は 0 から 255までの値しか取れないため、-1という値との比較が成立しなくなります。

(5) 棒グラフの表示

```
リスト1-2-5 bargraph.cと実行例
           <bdscio.h>
#include
main()
{
             d1,d2;
       int
       printf ("datal->");
                             /* input datal */
       scanf ("%d", &d1);
       printf ("data2->");
                             /* input data2 */
       scanf ("%d", &d2);
       bar (d1);
       bar (d2);
}
                             /* output bar & data */
bar(n)
       int
              n;
{
       int i;
       for (i=1; i<=n; i++) printf ("*");
       printf (" %d\n", n);
}
〈実行例〉
A>bargraph
data1->15
data2->20
****** 15
******* 20
A>
```

アスタリスク(*)を指定した数だけ画面に出力し、棒グラフを作るプログラムです。

ここで紹介したプログラムは BASIC などでも良く作成されるものですが、 実際のプログラミング作業は BASIC に比べ、難しくなります。しかしながら、その動作速度はさすが C 言語と言わしめるものがあります。

これらはすべて、ディスクのファイルアクセス処理のない簡単なサンプルです。 是非、すべてを理解して欲しいと思います。

第 **2** 章 Cのプログラムスタイル

第1章では全くの初心者の方がプログラムを作成する場合のサンプル をご紹介しました。C言語は非常にシンプルですが、細かい部分ではど うしてもテクニックを必要としますから、それが初心者の頭を悩ますと ころでもあります.

本章では、練習のためのサンプルプログラムではなく、本格的にBDS Cを使っていくために必要な部分を学習します。 ざっと読み流しても構 いませんが、判らないままにはせず、自分のものになるまで理解してく ださい、実際にプログラムを作成する場合、大きな力になるはずです。

2-1 ポインタの使い方

ポインタは小型で高速なオブジェクトを出力するC言語におけるスーパースターであり、非常に特徴的な機能です。

この機能のおかげでシステム記述言語として、ハードウェアに密着したソフトウェアやOS、言語などが作成しやすいと言われているわけです。しかし、その利用は必ずしも簡単とは言えません。

2-2, 2-3節で述べるコマンドラインの受け取り方や文字列の処理などでは、ポインタは使わないわけにはいかないのですが、ポインタは初心者程、無理に使おうとする傾向があり、使わなくても良い場合まで使いがちです。ポインタを利用するプログラムの多くは配列を用いて書き換えることが可能で、その方がプログラムの理解もたやすくなります。

ポインタの使い方に慣れてくると、ポインタを使うことでプログラムが小さく、作成も簡単になる場合と、複雑で混乱を招きやすくなる場合との区別がつくようになってきます。ポインタ、ポインタと呪文のように唱えるより、プログラムを早く、正確に作ることを最初は優先すべきだと思います。ただし、ポインタは重要な概念です。無理にポインタを使う事には反対ですが、ポインタを知らずに済ますことはできません。

(1) 文字列に対するポインタの使い方

さて、ポインタの具体的な使い方を説明するために、ポインタが最も多用される場合として、文字列の処理を扱ってみます。標準関数に strlen という文字列の長さをカウントする関数がありますが、これと同じものを配列を用いて実現した例をリスト 2-1-1に、ポインタを用いたものをリスト 2-1-2 に作成してみました。説明のため、関数名を前者は strlen1、後者は strlen2 としていますが、動作は全く同じです。

なお、文字列については2-4節で詳細に触れています。

リスト 2-1-1 strlen1.cと実行例

```
main()
{
       printf ("length=%d", strlen1("HELLO"));
}
strlen1(s)
       char s[];
{
       int
              i;
       i=0;
      while (s[i]) i++;
       return (i);
}
〈実行例〉
A>strlen1
length=5
A>
              リスト 2-1-2 strlen2.cと実行例
main()
{
       printf ("length=%d", strlen2("HELLO"));
}
strlen2(s)
       char *s;
{
             i;
       int
       i=0;
       while (*s++)
                     i++;
       return (i);
}
```

〈実行例〉

A>strlen2

length=5

A>

関数 strlen1, strlen2によって、"HELLO" という文字列の長さを求めます。 C 言語の配列では、配列の 0 番めの要素よりメモリの下位番地(0 に近い番地)から順に格納されています。

配列の0番めの要素	s [0]	(下位番地)
1	s [1]	
2	s [2]	
3	s [3]	
:	:	(上位番地)

C言語では必ずこのような配列が並ぶように定められているため、ポインタを使うことができるわけです.

この両者でまず異なるのは、引数 s の受け方です。strlen1 では文字列をchar (文字) 型変数配列とみなし、s をその配列アドレス(が格納されている変数) として宣言していますが、strlen2ではポインタ(アドレスを内容とする変数) として宣言しています。従って、プログラム中でも strlen2ではsの内容をダイナミックに変化させているのに対し、strlen1では定数として扱い、s の値は変化させていません。

利用する変数の数、アルゴリズム、行数も同じですし、動作も全く同じです。BASICに慣れた方なら、strlen1は問題無く理解できると思います。その上で両者を対応させ、異なる点を突き詰めていくと、ポインタの概念もお分りいただけるでしょう。

さて、プログラム中で実際に異なるのは、whileのある行だけです。プログ

ラムの考え方は全く同じですから,()の中にある,

s[i] *s++

が,内容としては基本的に同じものでなければなりません.

ここで、s[i] は配列ですから、s という配列のi 番めの要素です(実際には s という変数の内容を先頭アドレスとする文字配列のi 番めの要素)。最初に while の()の中が評価される時には、i は 0 ですから文字列の最初の文字を表します。while 文は()の中が0 でない時そのすぐ後の文を繰り返しますから、もし、そこが文字列の終端文字(0)でなければ、i をインクリメントします(i++)。

この部分が strlen2では *s++ となっています。この場合の*は演算子で、変数 s の内容をアドレスとする内容を読み出します。++はインクリメントの演算子で、読み出し操作と同時に s の内容を1増加させます。*は++よりも優先順位が低いので、*s の値(つまり文字列の中身そのもの)ではなく、変数 s の内容(アドレス)を増加させるのだということに注意してください。つまり、*s で内容を参照すると同時に、

s = s + 1;

を実行するのと同じです。従って、i=0の時、*s++も文字列の第 1 文字 めを表します。

もし、*s の値を増やすのなら、(*s)++ とし、優先順位を変更するための() がなければなりません。

このように、表現は全く異なるものの、動作としては全く同じ関数を配列、

ポインタという2つの考え方を用いて作ることができるわけです.

念のため、ここで気になるバイト数ですが、関数の部分だけで strlen 1 は 62バイト、strlen 2 は 60 バイトとなりました。速度は比較していませんが、 大きな差はないものと思われます。

(2) 整数に対するポインタの使いかた

int などの整数に対しては、char と比べるとポインタの利用はやや少なくなります。これは、もともと文字列がC言語ではchar 型の変数配列として定義されているため、文字列を扱う場合にはポインタが避けて通れないのに対し、整数に対しては使わなくてもプログラムが作れるからです。

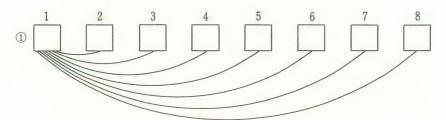
なお,整数配列も文字配列と同様,

配列の0番めの要素	i [0]	(下位番地)
1	i [1]	
2	i [2]	
3	i [3]	
:	:	(上位番地)

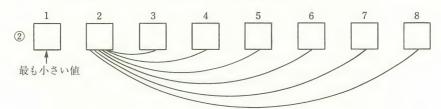
のように並びます。ただし、1つの要素は2バイトずつになります。

サンプルとして、バブルソートのプログラムを作ってみましょう。バブルソートはより効率の良いソート方法がいくつも考案されていますが、アルゴリズムが単純なため現在でも良く用いられます。

図2-1-3にその基本的なソートアルゴリズムを示します.



1と2~8を比較しながら1の方が大きければ値を入れかえる。 最終的に1には1~8の中で最も小さい値が残る。



2と3~8を比較しながら2の方が大きければ値を入れかえる。 最終的に2には2~8の中で最も小さい値が残る。

図2-1-3 バブルソートのアルゴリズム

1番から8番まで8個の数値があった場合、まず1番めと2~8番めの値とを順に比較していきます。その際、もし1番の値の方が小さければ1番とその数を入れ換えます。すべての数と比較し終ると、当然1番には最も小さい値が残っていることになります。

次に、2番めと3~8番の値を比較します。この時も同様の作業を行なうと、2番めには2~8番の数の中で最も小さい数が残ります。今度は3番と4~8番という具合にどんどん比較交換を繰り返していくと最終的には1番から8番までの小さい数字の順に並びます。

それでは、これを実際にプログラミングしてみましょう。数値を用意するのが面倒ですから、BDS Cで用意されている乱数発生用のrand 関数を用い、ランダムな100個の数値を発生させて配列 array に格納しておき、それをソートさせる事にしましょう。まず、リスト2-1-4 に配列を用いてソートする例を示します。

リスト2-1-4 bsort1.c

```
#include <bdscio.h>
#define ARMAX
                100
main()
{
        int
              i;
        int
                 array[ARMAX];
        for (i=0; i<ARMAX; i++)
                arrav[i]=rand ();
        bsortl (array, ARMAX);
        for (i=0; i<ARMAX; i++)
                printf ("%d\t", array[i]);
}
bsortl(arr,n)
        int
                 arr[];
        int
                 n;
{
                i,j;
        int
        int
                 tmp;
        for (j=0; j< n-1; j++)
                 for (i=j+1; i < n; i++)
                         if (arr[j]>arr[i])
                                  tmp = arr[i];
                                  arr[i] = arr[j]; arr[i]。の交換
                                  arr[i] = tmp;
                          }
}
```

数配列の先頭アドレス, n は配列要素の数です。なお、プログラムの先頭でARMAXという文字列に100を定義し、ソートする整数の数を100個にしています。ソート数を変える場合にはこの値を変更してください。

それでは次にこれをポインタを使って書き直した例をご紹介します。

リスト 2-1-5 bsort2.c

```
#include <bdscio.h>
#define ARMAX
                100
main()
        int
                i:
        int
                array[ARMAX];
        for (i=0: i < ARMAX: i++)
                array[i]=rand ();
        bsort2 (array, ARMAX);
        for (i=0; i<ARMAX; i++)
                 printf ("%d\t",array[i]);
}
bsort2(pnt,n)
        int
                *pnt;
        int
                n;
{
        int
                *spnt;
        int
                i, j;
        int
                 tmp;
        for (i=0: i < n-1: i++)
                 spnt=pnt+1; ←
                                       -spntにはすぐとなりの要素の
                                        アドレスをセット.
                 for (i=j+1; i < n; i++)
                         if (*pnt>*spnt)
```

```
tmp = *spnt;

*spnt = *pnt;

*spnt = tmp;

}

spnt++;

}

pnt++;

}
```

関数 bsort2 が変更した部分で、その他の部分は全く同じです。ポインタを 使うとどうしてもリストがわかりにくくなりますが、やむを得ないでしょう。

さて、関数 bsort 2 では比較変更する整数を指すポインタとして pnt と spnt を使っています。pnt は引数ですから、初期値として配列の先頭アドレスがあらかじめ設定されていますが、spnt はローカル変数ですから必ず値を セットする必要があります。pnt は基準位置、spnt は比較位置を指すポインタとして使っています。

pnt $\leftarrow 0$ 番めの整数のアドレス spnt $\leftarrow 1$ 番めの整数のアドレス

からスタートします。BDS C、 α -C では int は 2 バイトですから、spnt = pnt + 1 ではなく、spnt = pnt + 2 にしなければならないのではないかと思われる方もいると思います。確かにアセンブラでプログラムを作る場合は当然そうしなければ大きなバグを残してしまいますが、C 言語のポインタの加減算では、対象とする変数の1要素分が1になる事になっており、これが正しいのです。

16ビット、32ビット CPU を持つコンピュータでは int が必ずしも 2 バイトではなく、他のビット数を持つことも考えられるわけですから、移植性を考えると当然とも言えるわけです。

リスト2-1-4と2-1-5は完全に対応していますから、是非両者を見比べ、ポインタの感覚をつかんで頂きたいと思います。実行の結果を実行例2

-1-6に示します(両者の実行結果は全く同一です)

なお、気になるオブジェクトサイズですが、bsort1、bsort2の関数部分は それぞれ226バイト、204バイトとなりました。100個程度のデータでは計測 できませんが、実行時間にも差がでているでしょう。

なお、蛇足ですが、BDS Cには qsort というソート用の関数があり、大量のデータを高速に並べ変える必要がある場合にはこちらを用いた方が良いでしょう。 qsort はシェルソートアルゴリズムを用いています

実行例 2-1-6 バブルソートの実行結果

A>BSORT1

0	0	12	16	25	32	32	34	44	50
64	64	68	72	89	100	100	104	128	136
144	145	178	200	200	256	384	400	768	768
800	1650	1792	1792	1856	2262	2292	3301	3584	3840
3968	4257	4269	4293	4329	4414	5123	6602	6671	7168
7936	7936	8259	8282	8331	8402	8476	8757	8829	9231
10247	11769	12948	12984	13542	13599	14592	15872	15872	16406
16518	16548	16564	17209	17515	17658	18463	18512	18658	18945
19719	20494	20890	20999	22140	23283	25640	25713	25856	26115
26829	26883	27454	29184	29441	29697	30976	31232	31744	32000

A>

2-2 コマンドラインの受け取り方

C言語でプログラムを作成する際、その多くの場合にコマンドラインからの文字列を利用します。C言語ではこの受け取り方はほぼ固定されており、OSを問わず、移植性が高い部分です。

このサンプルプログラムはコマンドラインからの文字列を受け取り、そのまま画面に表示するものです。プログラム作成中に、特別な文字や記号などを引数としたい場合に、それが有効かどうか判らないことがあります。そのような時、リストのような簡単なプログラムは意外に役立ちます。

```
リスト2-2-1 args1.cと実行例
```

〈実行例〉

A>args1 MITARAI 31 ABCD 1 2 3

```
argc = 7
arg1="MITARAI"
arg2="31"
arg3="ABCD"
arg4="1"
arg5="2"
arg6="3"
```

さて、mainの関数宣言の部分に注目してみましょう。

```
main(argc, argv)
int argc;
char *argv[];
```

というのはコマンドラインからの引数(文字列)を受け取る際の定石です. main 関数は実行が開始される最初ですから、main を呼び出す上位の関数とい うものは存在しないわけですが、C言語では main 関数であろうとも他の関数と全く区別しませんので、システムが自動的にコマンドライン文字列を解析し、あたかも main に対する上位関数があるかのようにその結果を main 関数に渡すわけです。

ここで、argc、argvは

argc 引数の数

argv 引数文字列のポインタ変数配列の先頭アドレス

となります。要は、

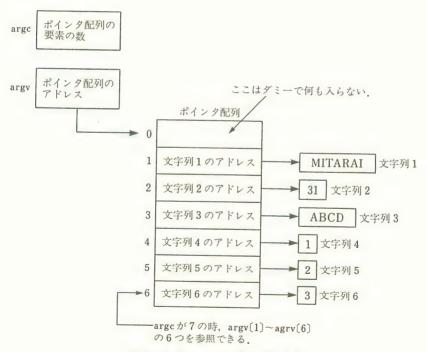


図 2-2-2 argv と文字列の関係

という構造になっており、2段階のメモリ参照で、文字列そのものを参照することができるわけです。実際のプログラムでは

argv[1] -1 つめの文字列の先頭アドレス argv[2] -2 つめの文字列の先頭アドレス : argv[n] -n 番めの文字列の先頭アドレス

となります。文字列そのものを表示したければ,

printf (argv[n]);

とするだけで、n番めの文字列を画面に出力できます。

この考え方は初心者の方にはなかなか難しいのですが、定石ですので、判らなければそのまま覚えてください。そのうちにメカニズムがはっきり判るようになると思います。なお、printf 関数は一般に、その第一引数は文字列をじかに""で囲んで引数とする場合が多いのですが、そのような記述も実はこの文字列そのものがプログラム中のどこかに埋め込まれ、printf 関数への実際の引数は文字列のアドレスとなっています。つまり、文字列を記述するという事は文字列への先頭アドレスを書くのと同義なのです。

なお、リスト2-2-1では

printf("arg%d=\frac{\psi}{s}\frac{\psi}{n}, i, argv[i]);

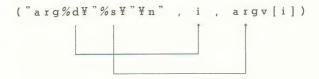
となっています。ここではprintf の重要な機能のいくつかを用いています。 まず、()の中を分解してみると、

- (a) "arg%d=\"%s\"\"n"
- (b)
- (c) argv[i]

の3つに分けられることが判ります。ここで(a)は文字列ですが、中にいくつか特殊な記号が入っています。すでに出てきたものもありますが、もう一度確認しておきましょう。

- %d %は制御文字で、%の直後にある文字によりさまざまな機能を持たせることができます。dは10進整数の指定で、文字列の後にある引数(この場合はi)の値を10進整数に変換して表示します。
- %s %の後が s の場合, 引数の値を文字列へのポインタ(文字列の先頭アドレス)と解釈し, その文字列を表示します. この場合は argv [i], すなわち i 番めのコマンドラインの文字列となります.

これらの%は文字列の中にそのまま組み込まれて、printf 文実行の際に変換されて表示されます。この時、制御文字列と引数の関係は、



となり、制御文字がでてくるたびに次々と新たな引数を参照していくことになります。このように printf は引数の数が一定しない特殊な関数で、文字列の内容と引数の数を対応させなければ正しく動作しません。

なお、文字列中にでてくる¥は次のような意味を持っています。

- ¥" 文字列はその最初と最後を"で囲みますから,"自身を文字 列中に含ませることができません。そこで文字列では¥記号 を特殊記号として扱い、¥"は文字"そのものとして扱い ます。
- ¥n 改行を示します.

これらの¥を伴うものは、¥の文字そのものが残るわけではなく、コンパイルされた時点で変換されて目的のコードに変わります。つまり、これはprintf 文の機能ではなく、コンパイラの機能になるわけです。

さて、このプログラムは簡単ですが、基本的な C 言語の機能をたくさん含んでおり、非常に良いサンプルだと思います。特に初心者の方は判るまでテストしてみることをおすすめします。

さて、main 関数の引数として一般にはリスト 2-2-1 のように宣言する場合が多いのですが、argv をポインタ変数として宣言することがあります。機能としては全く同じなのですが、考え方が異なります。

- (1) main(argc, argv)
 int argc;
 char *argv[];
- (2) main(argc, argv)
 int argc;
 char **argv;

(1)は文字列へのポインタ変数配列の先頭アドレスが argv (の内容) である という解釈をする場合, (2)は argv というポインタ変数の初期値がポインタ 変数配列の先頭に設定されていると解釈する場合です。

リスト 2-2-3 に (2) の方法を用いてリスト 2-2-1 を書き直したものを示しますが、少しプログラムが変わっているのに気付かれたと思います。ポインタ配列へのポインタとして宣言した場合、argv はポインタ変数として扱う方が自然になります。

なお、*++argvはargvの内容をインクリメントした後、その内容をアドレスとした内容になります。

リスト2-2-3 args2.c

コマンド引数の受取りは非常に重要な部分で、慣れれば簡単なのですが、 ポインタ配列というやや高度な概念を含むため、初心者の方は意外に悩むこ とが多いものです。本項でしっかり把握してください。

2-3 ファイルの入出力

さて、ファイルの入出力はいくつもの標準関数に支えられて、大変便利になっています。BASICなどに較べるとはるかに機能も豊富ですが、そのぶん関数の種類が多く、どれを使えば良いか悩むこともしばしばです。

1つ1つの関数について、マニュアルを良く読むことも必要ですが、まず基本的ないくつかの関数をしっかり理解することも重要です。

〈2種類のファイル IO〉

一般的なC言語には低レベルファイルIO(Input Output = 入出力)とバッファードファイルIOという2つのファイル入出力用の関数群が用意されています。これらは基本的には次のように考えれば良いでしょう。

低レベル IO - ファイルの物理的な記録単位(CP/Mでは 128 バイト)での入出力を行なう場合

バッファード IO — ファイルと 1 文字単位でデータの出し入れを行なう場合

多くの場合、バッファードファイル IO の方が操作は簡単ですが、バッファードと言えどもその中で低レベルファイル IO を利用しているのですから、処理そのものの速度はどうしても遅くなります。ファイルコピーのようにどちらを利用してもできるようなプログラムの場合には当然低レベル IO の方が処理速度は速くなります。

〈低レベルファイル IO〉

ファイルをブロック単位で扱う場合に使いますので、ファイルのランダム アクセス、ブロック転送などに有効です。

低レベルファイル IO として重要なのは次の4つの関数です。

open — 既にディスク中に存在するファイルを読み書きするため にファイルをオープンします。

creat — ファイルがある場合には消去し、書き込みのために新し くファイルをオープンします。

close - ファイルをクローズします。

read 一ファイルをブロック単位で読み出します。

write 一ファイルにブロック単位で書き込みます。

seek — ランダムアクセスなどのため、ファイルの読み書きポインタを変更します。

まず、openが既に存在するファイルをオープンし、読み出し、書き込み、あるいはその双方に備えるのに対し、creatはファイルがある場合には消去してから書き込み用にオープンします。従って、creatでは同じ名前のファイルは強制的に消去されますから、注意が必要です。open、creatによってオープンされ、書き込み処理が終了したファイルは必ず最後に close によってクロ

ーズしなければなりません.

read, write はそれぞれファイルからの読み込み,ファイルへの書き込みを行ないます。BDS Cではこの単位は128バイトになっており、1バイトだけが必要な場合にも128バイトいっぺんに読み書きする必要があります。

一応,これらの関数の存在と簡単な機能だけを知っておけば、後はプログラム作成時にマニュアル通りに記述していくだけでも何とかなります。

まず、簡単な例として、ファイルの内容を16進数とアスキーの両方で表示するプログラムを作成してみます。

リスト 2-3-1 cdump. c ファイルダンププログラム

```
#include
 1:
                      <bdscio, h>
 2:
 3:
      #define
                       ERROR
                              -1
 4:
      #define
                       INPUT
5:
 6:
      main (argc, argv)
7:
               int
                       argc;
8:
               char
                       *argv[];
9:
      {
10:
                       buf[128];
                                       /* sector buffer */
              char
11:
               int
                       fd;
12:
                                        /* size of sectors */
               int
                       SZ;
13:
              unsigned adr:
14:
15:
             if (argc!=2)
16:
17:
                       printf ("Usage: cdump filename");
18:
                       exit ():
19:
              }
20:
21:
              fd = open (argv[1], INPUT);
22:
              if (fd == ERROR)
23:
24:
                       printf ("Can't open %s", argv[1]);
25:
                      exit ();
26:
              }
27:
28:
              adr = 0;
29:
              while (1)
30:
31:
                       sz = read (fd.buf.1):
32:
                       if (sz < 1)
                                       break; /* file end */
```

```
dump (buf, adr);
33:
                       adr += 128;
34:
35:
              close (fd);
36:
37:
      }
38:
39:
      dump (buf, adr)
40:
41:
              char
                      *buf:
42:
               unsigned adr;
43:
44:
                       i.i:
               for (i=0; i<128; i+=16)
45:
46:
                       printf ("%04x ",adr+i);
                                                        /* address output */
47:
48:
                                                        /* hex output */
                       for (j=0; j<16; j++)
49:
                                printf ("%02x ", buf[i+j]);
50:
51:
52:
                       printf (" ");
53:
                       for (j=0; j<16; j++)
                                                         /* ascii output */
54:
55:
                                                         printf (".");
56:
                                if (buf[i+j]<' ')
                                                         printf ("%c", buf[i+j]);
57:
                                else
58:
                       printf ("\n"):
59:
60:
61:
     }
62:
```

[15行~19行]

プログラムの実行方法が分らなくなったり、誤った指定をした場合に、CDUMPの使用方法を示す部分です。15行めで(argc!=2)となっています。argcが2以外の場合、つまりファイル名が指定されなかったり、あるいは2つ以上の文字列をプログラム名に続けて指定してしまった場合はすべてこのUsage:を表示して終りになります。!=は一致しない時に真となる比較演算子です。

exit()はプログラムを中断し、強制的に終了する場合に使います。

[21行~26行]

関数 open を実行し、ファイルが存在しないなどの理由でエラーとなった場合にはそのメッセージを表示して終ります。

[28行]

ダンプアドレスを初期化します。これはダンプの際、左端に表示するアドレスが0からスタートするように指定します。

[29行~35行]

ここがプログラムの本体部分です。31行でファイルをリードし、もし実際に読み込んだセクタ数(ブロック数)が指定した読み込みセクタ数の1より小さかった場合、ファイルの終了と考えられます。その場合にはこのループから抜け出します。

もし、正しく読み込めた場合にはその内容を関数 dump に送ってダンプし、 ダンプアドレス adr を 128 カウントアップしておきます。

[36行]

終了ですので、ファイルをクローズします。

実行例 2-3-2 cdump.cの実行例

A>cdump

Usage: cdump filename

A>cdump cdump.c

23 69 6E 63 6C 75 64 65 09 3C 62 64 73 63 69 6F 0000 #include. <bdscio 0010 2E 68 3E 0D 0A 0D 0A 23 64 65 66 69 6E 65 09 09 .h>....#define.. 0020 45 52 52 4F 52 09 2D 31 0D 0A 23 64 65 66 69 6E ERROR. -1., #defin 0030 65 09 09 49 4E 50 55 54 09 30 0D 0A 0D 0A 6D 61 e. . INPUT. O. . . . ma 69 6E 28 61 72 67 63 2C 61 72 67 76 29 0D 0A 09 0040 in (argc, argv)... 0050 69 6E 74 09 61 72 67 63 3B 0D 0A 09 63 68 61 72 int.argc;...char 0060 09 2A 61 72 67 76 5B 5D 3B 0D 0A 7B 0D 0A 09 63 .*argv[];...{...c (後略)

A>

〈バッファードファイル IO〉

さて、低レベルファイル IO 関数がディスクファイルの内容をブロック単位

でしか扱えないのに対し、バッファードファイル IO では 1 文字単位でファイルを扱うことができます。しかしながら、CP/M ではあくまでディスクとのやりとりは 128バイトという固定されたサイズでしかできないのだという事は知っておく必要があります。バッファード(Buffered)という意味もまさにそこにあり、1 文字入力の場合であればディスクからブロック単位のデータを読み取って一旦バッファー(メモリ)に蓄え、そこから1 文字1 文字切り取ってくることを示しています。

高級なファイルIOですから、使い方はかえってやさしいのですが、まずどのような関数があるか説明します。

fopen — 既に作成されているファイルをオープンします.

fcreat — 同一名のファイルがある場合にはそのファイルを消去し、書き込むためにファイルをオープンします.

fclose - ファイルをクローズします。

getc - ファイルから1文字読み込みます.

putc - ファイルに1文字書き込みます。

fgets - ファイルから1行読み込みます。

fputs - ファイルに1行書き込みます.

f printf — ファイルへの出力に対し、printf と同じ機能を持つ 関数です。

fscanf — ファイルからの入力に対し、scanfと同じ機能を持つ関数です。

この他にもいくつか関数がサポートされていますが、重要なものは上記の 9つで、これらをうまく使うことでほとんどの処理が可能です。

さて、バッファードファイル IO の考え方で一番重要なのは、文字が中心に 考えられている事です。つまり、上記の関数はすべて、英数字などを格納し たアスキーファイルを扱う事を前提に考えられており、COM ファイルなどの バイナリファイルを扱う際には必ずしも便利ではありません。

例えば、アスキーファイルではそのファイルの終端文字として、コントロールZ(1AH)が用いられますが、この文字が現れるとファイルの終了とみな

す関数 (fgets) とそのまま 1AH として返す関数 (getc など) がありますから、使用する際には注意が必要です。

さて、このバッファード IO 関数を使う場合、必ず bdscio. h という標準へッダーファイルを先頭でインクルードしなければなりません。bdscio. h の中ではこれらの関数で読み書きのために使うバッファーを確保する構造体を宣言していますので、これにより、バッファード IO 関数を利用する時には、

FILE iobuf;

などと宣言すれば自動的にバッファードファイル IO 用のバッファー,及びその状態などを記憶しておくフラグ・カウンタなどのメモリが確保されます。何が何バイトで、といった事を考える必要はありません。

リスト2-3-3はディスクからファイルを読み、それを行ごとにナンバーを付けて別ファイルとして格納するプログラムです。コマンドラインから、

A>fnum filel file2

とすると、file1の内容に行番号を付加し、file2として出力します.

入力のためにファイルからの行入力関数 fgets,出力に書式つきファイル出力関数 fprintf を利用している点に注意してください。

リスト 2-3-3 fnum. c 行番号付加プログラム

```
1:
      #include
                     <bdscio.h>
 2:
3:
     FILE
              fpin:
      FILE
 4:
              fpout:
5:
6:
    main (argc, argv)
7:
              int
                      argc;
8:
              char
                      *argv[]:
9:
    {
10:
              char
                      buf[200]:
11:
              int
                      lcnt:
12:
```

```
if (argc != 3) printerr ("Usage: fnum infile outfile");
13:
14:
                                                      printerr ("Can't open."):
              if (ERROR == fopen (argv[1], fpin))
15:
                                                      printerr ("Can't create.");
16:
              if (ERROR == fcreat(argv[2], fpout))
17:
              lcnt = 1:
18:
19:
              while (fgets (buf, fpin))
20:
21:
                       if (ERROR == fprintf (fpout, "%5d:\timestaks", lcnt, buf))
22:
                               printerr ("Write error,");
23:
                       lcnt++:
24:
25:
               fclose (fpin):
26:
               fclose (fpout);
27:
28:
     printerr (msg)
29:
30:
              char
                       *msg:
31:
      {
               printf (msg):
32:
33:
              exit ():
34:
      }
35:
```

[3, 4行]

ここでバッファード IO 関数を用いるための入出力用ファイルを宣言しておきます。ここでは入力用として fpin、出力用として fpout の2つを宣言していますが、このように入出力のどちらにも同じ書式です。なお、この宣言は関数の外側で行なわれていますので、どの関数からも自由にアクセスできる外部変数になります。関数の内部で宣言すればローカル変数として、その関数の内部でだけ利用可能となります。

[10, 11行]

10行めでは1行を格納するための行バッファーを200バイト宣言しています。fgetsでは1行の長さをチェックしませんので、もしこれより長い行がファイル中に存在すると暴走する可能性があります。一般にはこれより長いものはめったにありませんが、信頼性が特に要求されるような用途にはもっと確実な方法を選ぶ必要があります。

11行で宣言している lent は行カウンタです。 1 行読むごとにインクリメントされます。

[13行]

13行めで行なっているのはプログラム起動時のコマンドラインからの引数チェックです。fnum.c は必ず2つのファイル名を指定する必要がありますから、もしそれ以外(すなわち argc が 3 以外)であったら、Usage:以下を表示して終了します。printerr はそのためのメッセージを表示してプログラムを終了する関数です。

[15, 16行]

15、16行ではファイルの入出力のため、2つのファイルをオープンします。 fpin を入力用に、fpout を出力用にしています。ここで、ERROR というのは定数で、-1 を示しています。-般にこのような定数は # define # で宣言しなければなりませんが、非常に一般的な定数は # bdscio.# hの中で宣言されているので、宣言しなくても利用できます。このような定数には他に、

```
NULL 0

EOF -1

OK 0

TRUE 1

FALSE 0

CPMEOF 0xla(コントロールZのアスキーコード)
```

などがあります。

[18行]

行カウンタ lcntの値を1に初期化します.

[19行]

fgets 関数により fpin で指定されているファイルから1行を読み込み,buf へ格納します.かつもしその戻り値が0以外の場合には $\{\}$ の中を繰り返します.fgets はCR、LF(改行コード)があるとそれをLF(Yn)だけに置き直

し、かつ最後にC言語での文字列終端文字として0を追加します。

なお、ファイルが終了した場合、fgets は 0 を返します。while の条件は 0 以外(真)の場合ループし、0 (偽)の場合、ループを終了しますので、このような場合には都合が良くなります。

[21, 22行]

プログラムの本体部分です。fgets で読み込んだ1行に行番号を付加して,fpoutのファイルに出力します。fprintf は書き込みエラー時に-1を返しますから、ここでERRORと比較し、エラーなら強制終了します。

fprintf などは良く ERROR チェックなしでも利用されることがあり、そのような場合にはファイルの書込みチェックはされません。これは書き込む位置が1箇所でなく複数の箇所になる場合など、プログラムサイズが大きくなり、動作速度も遅くなりますので、意識して省かれるわけです。しかし、書き込みエラーはしばしば発生するトラブルであり、やはりエラーチェックはすべきでしょう。

[23行]

ラインカウンタをインクリメントします.

[25, 26行]

プログラムの終了処理です。入力用ファイルfpin,および出力用ファイルfpoutをクローズします。

[29~34行]

文字列を表示してプログラムを強制終了する関数 printerr です。

実行例 2-3-4 fnum, cの実行例

A>fnum

Usage: fnum infile outfile

A>fnum fnum.c fnum.n

A>type fnum.n

1: #include <bdscio.h>
2:
3: FILE fpin;
4: FILE fpout;
5:
6: main(argc, argy)

(後略)

A>

2-4 文字列の処理

BDS Cに限らず、C言語では文字列の処理がBASICなどと異なるので、最初のうちはとまどう事もしばしばです。BASICでは文字変数は文字列への先頭アドレスと文字数という2つの要素で記憶されている場合が多いのですが、Cではあくまで文字列は文字型(char)の配列としてのみ扱われ、文字列の最後に0(16進のゼロ)を置き、この終端文字と先頭アドレスで扱います。

この方法は文字列をアドレス(2バイト)だけで受け渡しできる点で有利ですが、文字列操作では必ずしも便利ではなく、文字列の先頭何文字かを取り出したりする場合には文字を1字ずつ追いかけなければならないため実行が遅くなるという欠点も持っています。また、文字列の中に0(16進のゼロ)を使うことはできませんから、プログラムオブジェクトなどのバイナリデータを扱う場合には特に不便です。一般にC言語でバイナリデータを扱うプログラムを作成する場合、自分でいくつかの文字列操作用関数を作る必要があ

ると考えてよいでしょう。本項での文字列操作とは一般の文字や文章の操作 であると考えてください。

〈文字列処理の基本ファンクション〉

文字列をそのまま別の場所に移したり、付け足したり、比較したりする場合は非常に多いものです。これらには次のような関数を利用します。

(1) strcpy (string copy)

文字通り、文字列をコピーする場合に用います.

strcpy (t,s)

とした場合、sの文字列をtにコピーします。この場合、tもsも文字列へのアドレスで、tはsをコピーできるだけの十分な長さを持っている必要があります。

(2) strcat (string concatenate(文字列連鎖))

文字列を連結する場合に使います.

strcat (s,t)

とした場合、sの文字列の後にtを付加します。sはtが後に付加されるだけの十分な長さを持っている必要があります。

リスト 2-4-1 string.cと実行例

#include <bdscio.h>

main(argc, argv)
 int argc;
 char *argv[];
{

char buf[100];

```
strcpy (buf,argv[1]);
strcat (buf,argv[2]);
printf (buf);
}
```

〈実行例〉

A>string ABC DEF ABCDEF

A>string JKLMN OPQRST JKLMNOPQRST

A>

リスト 2-4-1は、コマンドラインの入力した 2 つの文字列を合成して表示します。 2 つ入力がない場合のチェックは行なっておりませんので必ず、string XXXX YYYY というフォーマットで実行してください。

(3) strcmp (string compare)

文字列同士を比較し,同一かどうかをチェックします。

```
strcmp (sl,s2)
```

とした場合、一致すれば0、一致せず最初に異なった文字について、s1>s2なら正の値、s1<s2なら負の値を返します。

リスト 2-4-2 strcmp. c と実行例

printf ("not same");
else
printf ("same");
}

〈実行例〉

A>strcmp ABCDEFG ABCDEFG same

A>strcmp ABCDEFG ABCDEF not same

A>

リスト2-4-2は、コマンドラインから入力した2つの文字列が等しいかどうかをチェックし、等しければ "same"、同じでなければ "not same" と表示します。

(4) strlen (string length)

文字列の長さを求めます、終端文字の 0 は含みません。

この関数はポインタの項で解説した strlen 2 と全く同じものですので、リスト 2-1-1、 2-1-2 を参照してください。

これらの関数は簡単なもので、自作してもほとんど数行でできるようなものばかりですが、非常に重要です。これらを使ってみるのが、文字列処理をマスターする第1歩でしょう。これら以外にもいくつかの関数がありますが、その使用瀬度は上記の4関数程高くはありません。

〈ユーティリティ関数〉

文字列処理関数は非常に便利なものですが、BASICに慣れたかたなどは、 かなり物足りなく感じる場合も多いものです。そこで、標準関数にないBASIC ライクな関数をいくつか紹介しておきます。文字列というものがどう扱われ、 どのようにすれば文字列処理が可能なのか良いサンプルになると思います。 特にここで紹介する関数が便利なのは、BASICのプログラムをCに移植する 場合です。

```
char *left(str,len)
  char *str;
  int len;
```

BASICのLEFT\$関数に対応する関数です。strに文字列へのアドレスをセットし、lenに文字列の先頭からの長さを指定すると、文字列 str がその文字数までで打ち切られ、str のアドレスが戻り値として返ります。元の文字列は失われることになりますから注意してください。

なお、指定の長さより与えられた文字列の方が短ければそのまま返ります.

リスト 2-4-3 left.cと実行例

```
#include
                <bdscio.h>
main (argc, argv)
        int
               argc:
        char
                *argv[]:
{
        printf (left (argv[1],3));
}
                                -左から3文字を切り取る。
left(str,len)
        char
                *str:
        int
                len:
{
        char
                *S;
        s = str:
        while (len--)
                if (!*s++) return (str);
```

〈実行例〉

A>left ABCDEFG ABC

A>left JKLMNOP JKL

A>

サンプルプログラムでは、コマンドラインから入力した文字列を先頭3文字で切り取り、表示します。

```
char *right(str,len)
  char *str;
  int len;
```

BASICの RIGHT \$関数に対応する関数です。str に文字列へのアドレスをセットし、len に文字列最後からの文字数をセットするとその位置のアドレスを戻り値として返します。文字列の途中から使うことになりますので、元の文字列は失われません。

指定の長さより文字列が短かければ文字列はそのまま戻ります.

リスト 2-4-4 right.c と実行例

#include <bdscio.h>

main(argc, argv)
int argc;

char *argv[];

```
{
        printf (right (argv[1],3));
}
                              -右から3文字切り取る。
right(str, len)
        char
              *str:
        int
               len:
{
        int
        char *s;
        s = str:
        i = 0:
        while (*s++)
                       i++;
        if (i<len)
                       return(str);
        while (len--)
                       S--;
        return (--s);
}
〈実行例〉
A>right ABCDEFG
EFG
A>right JKLMNOP
NOP
A>
```

リスト2-4-4はコマンドライン引数の文字列の右3文字を切り取ります。

```
char *mid(str,pos,len)
  char *str;
  int pos;
  int len;
```

BASIC の MID \$ 関数に対応する関数です。str に文字列へのアドレスをセットし、pos に切り取りたい文字の先頭からの位置、len に長さを指定すると、目的の文字列へのアドレスが戻り値として返ります。元の文字列は失われます。

なお、指定の位置より文字列が短ければヌルストリング(終端文字だけで 内容のない文字列)が返り、指定の長さより文字列が短かければ、途中で打 ち切られた文字列となります。

リスト 2-4-5 mid. c と実行例

```
#include <bdscio.h>
main (argc, argv)
        int argc:
        char *argv[];
{
       printf (mid (argv[1], 2, 3));
}
                             -2文字めから3文字切り取る。
mid(str, pos, len)
        char
              *str:
        int
              pos, len;
1
        int
              i:
        char
              *S;
        i = 0:
        while (pos--)
               if (!*str++) return (--str);
        s = --str:
       while (len--)
               if (!*s++) return (str):
        *s = '\\1':
        return (str):
}
```

〈実行例〉

A>mid ABCDEFG BCD

A>mid JKLMNOP KLM

A>

リスト 2-4-5 はコマンドライン文字列の 2 文字めから 3 文字分を切り取って表示します。

2-5 構造体の使い方

構造体もC言語の特徴の1つです。BASICなどには無い概念なので、最初はとまどいますが、決して難しいものではなく、むしろプログラムを読みやすく、判りやすくしてくれます。

ここではサンプルとして、役には立ちませんが簡単な電話帳プログラムを 作成してみましょう。名前・年齢・電話番号の3つがデータで、2人分のデータが入力されるとその内容を表示して終了します。

このプログラムは構造体を使わなくても実現できますので、比較の意味で まず配列で作成したものを説明します。

リスト 2-5-1 memlist1.c と実行例

```
for (i=0; i<MEMBERS; i++)
{
    printf ("name->");
    gets (name[i]);
    printf ("age ->");
    scanf ("%d", &age[i]);
    printf ("tel ->");
    gets (tel[i]);
}
for (i=0; i<MEMBERS; i++)
{
    printf ("%s (%d) tel:%s\fm", name[i], age[i], tel[i]);
}
</pre>
```

〈実行例〉

A>MEMLIST1

```
\begin{array}{l} {\rm name} -> \underline{{\rm T.\,MI\,TARA\,I}} \\ {\rm age} \ -> \underline{{\rm 31}} \\ {\rm tel} \ -> \underline{{\rm 045-911-7427}} \\ {\rm name} -> \underline{{\rm R.\,MI\,TARA\,I}} \\ {\rm age} \ -> \underline{{\rm 26}} \\ {\rm tel} \ -> \underline{{\rm 045-911-7427}} \\ \end{array}
```

T.MITARAI (31) tel:045-911-7427 R.MITARAI (26) tel:045-911-7427

A>

判りにくい部分は全くないと思います。名前を入れる配列 name, 年齢を入れる配列 age, 電話番号を入れる配列 telの3つを用意します。勿論 name, telは文字列ですから文字列の配列で2次元配列となります。

これらの配列に対し、文字列の行入力関数 gets と入力関数の scanf を用いて値をキーボードから入力し、2人分入れるとその内容を画面に出力します。人数は MEMBERS で、2が指定されていますからこの値を変更すれば何人にでも対応させることができます。

このようなプログラムは BASIC などでは常套手段であり、配列要素の順

に、関連のあるデータを格納する方法は良く用いられます。しかし、ここで name、age、telというのは相互に関係の深い配列であるにも拘わらず、完全 に独立しているため、その関係が明確ではありません。そこで C 言語ではこの 3 つの配列をひとまとめにし、name[]、age、tel[]という型の異なる複数の要素を一つの要素として扱えるようになっています。これを構造体といいます。

構造体を用いて書き直したものをリスト2-5-2に示します。

リスト 2-5-2 memlist 2. c

```
#include <hdscio.h>
#define MEMBERS 2
struct member
        {
                char
                         name [25]:
                char
                         age:
                char
                         tel[15];
       };
main()
1
        struct member
                        list[MEMBERS]:
        int
                i:
        for (i=0: i<MEMBERS: i++)
                printf ("name->");
                gets (list[i].name);
                printf ("age ->");
                scanf ("%d", &list[i].age);
                printf ("tel ->");
                gets (list[i], tel);
        for (i=0; i<MEMBERS; i++)
                printf ("%s (%d) tel:%s\n", list[i], name, list[i], age, list[i], tel);
        }
}
```

struct から main () の前までが、構造体タグ member の定義であり、以降

```
struct member list;
```

とするだけで、member 型の要素を持った構造体の宣言ができます。例のようにさらにそれを配列にしておけば、構造がはるかに明確に指定できます。

プログラムそのものは長い文字列を書くことになりますので、ちょっと見ずらく感じられますが、リストが長くなってくるとはるかに判りやすくなります。

このように複数の配列を対応させて扱うような用途には、構造体は非常に 便利なものです。

なお、構造体の要素への読み書きは

list.name : 構造体変数 list の要素 name list.age : 構造体変数 list の要素 age list.tel : 構造体変数 list の要素 tel

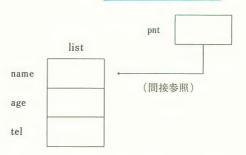
のように、変数名と要素名の間をピリオドでつなぎます。ポインタにより間 接的にアクセスする場合には、

struct member *pnt; pnt=list;

などとしておいて,

 $pnt \rightarrow name$ $pnt \rightarrow age$ $pnt \rightarrow tel$

のように-と>の2つの文字で結合させます。このポインタによる間接アクセスは混乱を招きやすいのですが、次のように考えれば良いでしょう。



図のように pnt という変数の内容 (アドレス) が指しているのが member 型の (list という名前の) 構造体です。list というのは名前であると同時にこの構造 体の先頭アドレスを示す定数です。つまり、直接に定数から読み書きする場合には、

list name

とすれば良く,間接的に(ポインタ)変数の内容から参照する場合には

$pnt \rightarrow name$

となるわけです。ところが文字列の受け渡しなどでは、19ページのリスト 2-1-1などのように引数が配列(の先頭アドレス)であることを宣言する と、その引数については間接的な読み書きにもかかわらず

s [i]

などのようにあたかも直接的な読み書きであるかのように書くことができますが、構造体ではこのような記述ができません。初心者が誤りやすいポイントですので、構造体の場合は直接参照か間接参照かを意識し、プログラムを書き分けるようにしてください。

配列に比べると、構造体は特殊な操作をするような印象を与えますが、よく考えてみると関係のあるものを集めてブロック単位で扱う方がはるかに自然で合理的であることが納得できると思います。

第 3 章

BDS C特有の問題点

本章はBDS C. α -Cにおいて、問題になりやすい部分について解説します。その多くはこれらのコンパイラが C 言語のサブセットであることから起るものです。

ただ、これらの欠点があるからBDS C、 α -C は良くないという結論は 短絡的で、どのようなコンパイラでも固有の問題というものはあるもの です。これらの欠点のため、コンパイルが速い、バグが少ないという、大きなメリットを忘れてはならないと思います。

なお、本章のような内容を読まれて、「このような機能削除版のコンパイラで果して正しい C 言語の学習ができるのだろうか」と思われるかも知れません。あらかじめ断言しておきますが、BDS C、 α -Cで覚え、経験した C 言語のノウハウが他の C コンパイラで役に立たないということは決してありません。ご安心ください。

3-1 コンパイルエラー

BASIC は言語仕様上エラーの検出がたやすく、有名な Syntax error から始まり、多くの誤りを完全に指摘してくれます。ところが、 C言語はフリーフォーマットであるため、行というプログラム分割単位を持たず、 コンパイル時に起るエラーがどこでどのようにして発生したか判定しにくいという欠点があります。

エラーメッセージとして最も頻出するのは文末にセミコロン (;) をつけ 忘れる

Missing semicolon

だと思います。これは既に読者の方々も経験ずみなのではないでしょうか. また、エラーメッセージが予想以上に多く発生することもしばしばあります。これは大抵の場合、はじめにあるエラーが連鎖的に後のエラーを招いたもので、多くは最初のエラーを取り除くと消えます。

リスト3-1-1 大量のエラーが発生する例

```
1: #include <bdscio.h>
2:
3: main()
         ---ここにカーリーブレイス | が落ちている。
           int
5:
                   i,s;
            char f:
6:
7:
           for (s=2; s<=100; s++)
8:
9:
                    f = 0:
10:
                    for (i=2; i <= s-1; i++)
11:
12:
                            if (s\%i == 0)
13:
14:
```

〈コンパイル例〉

A>cc f:error1

BD Software C Compiler v1.50a (part I)

5: Missing from formal parameter list: i カーリープレイス | が
 5: Missing from formal parameter list: s
 6: Missing from formal parameter list: f
 21: Unmatched right brace → | | | が対応せず | が1つ多いため

21: Unmatched right brace → | | が対応せず|が1つ多いため 21行めでエラーとなる。 これと 5、6 行めのエラーを考慮 すると 4 行めのミスは容易に判明する。

エラーの対処方法は場合、場合によって異なり、その都度エラーメッセージを解析していくことになりますが、基本的には次のような誤りが多いと思われます。

(1) {と} のアンバランス

C言語では { が出現するたびに字下げを行ない、 { により、もとの桁位置に戻すようにすると、 { } のバランスが崩れるのを防ぐことができます.これについては本書のリストのほとんどが実施していますので、参考にしてください.

(2) (と)のアンバランス

これは BASIC でも同じです。複雑な式などではどうしても誤りやすいので数えてみた方が良いでしょう。なお、

if ((kansuul () && flagl)||(kansuu2 () && flage2))

などと複雑な条件式などでは、フリーフォーマットである事を積極的に利用 して、

のように条件を複数の行に分けて記述することができます。このように記述 すると単純にコンパイルエラーの発生を防げるだけでなく、ロジックがわか りやすくなるという特徴もありますから、試してみると良いでしょう。

(3) タイプミス

if, for などは短いのでミススペルをすることはまずありませんが、while, switch, default などは誤りやすいので気をつけてください。誤って覚えていたりすると意外と悩むものです。

(4) 文字列のダブルクォーテーションマーク

文字列は必ず""で囲まなければなりませんが、最後の"を忘れることが あります、これは通常

String too long (or missing quote)

というメッセージが出るため、すぐに修正することができますが、#if、#endifなどで条件コンパイルを指定しているとエラー発生行を誤って表示する場合があります。

リスト3-1-2 string too long エラーがでない場合

1: #include <bdscio.h>

2:

3: #define DEBUG 0

4:

5: main()

こに文字列の終了を示す

```
6: {
                                             ダブルクォーテーション"が
                 printf ("it is in main().); 落ちている.
     7:
     8:
                 test ():
     9: }
    10:
    11: test()
    12: {
    13: #if DEBUG
    14:
                 printf ("it is in test()"):
    15: #endif
    16: }
〈コンパイル例〉
A>cc f:error2
BD Software C Compiler v1.50a (part I)
 13: Conditional expr bad or beyond implemented subset
                               このエラーメッセージは13行めの#ifの使い方が
A>
                               正しくないという指摘。
                               しかし、それは正しくない。このエラーは7行めの"
不足のため引き起こされたもの。
                               このエラーメッセージはα-Cでも同様に出現する.
```

10行めで落した"が100行めの#ifでエラーを発生させた場合など、非常に苦しむことになります。もし#ifで発生したエラーがあったら、文字列の "を落していないかチェックしてください。

コンパイルエラーは最初は非常に多いものですが、慣れるとその発生瀕度も減ってきますし、修正の要領も判ってきます。99%のエラーはそのメッセージによってどういうものか判定できますから、少しずつエラーを減らすようにしていけば良いでしょう。

3-2 BDS Cで削除されている機能

BDS C および α -C は C 言語の仕様を一部削除したサブセット版です。通常の使用ではあまり問題になることはないのですが、本質的な仕様の点で次

のような制約がありますから、注意しなければなりません。

- (1) 実数 (float), 倍精度整数 (long), 倍精度実数 (double) が使えない.
- (2) スタティック変数がない。
- (3) 変数の初期値設定機能(イニシャライザ)がない。

BDS Cでは実数、倍精度整数については関数パッケージ(float, long)が用意されており、関数の形で用いることができます (α -Cには付属していません)。しかし、他のCコンパイラ用に作成されたプログラムなどをこの関数を用いてBDS C に移植するのはかなり大変な作業です。float 関数の具体的な使い方については、3-4項をご覧ください。

(2), (3)は C 言語の本質的な部分であり、非常に残念なことです。 BDS C は完成されたオブジェクトを ROM などに焼き込み、制御機器などへの組み込み用ソフトウェアを作成することができますから、この機能を付加することが難しく、削除されているのだと思います。

しばしば、C言語のサンプルとして、次のようなものがあります。

リスト3-2-1 イニシャライザを用いたプログラムとコンパイル例

```
1: #include <bdscio.h>
2:
3: main()
                                    intのデフォルトが自動変数 (auto)
                                    になっている場合、どのようなコ
4: {
                                           の行はエラーになる
5:
                      a = 100:
           int
           static int b = 200;
                                   -スタティック変数と明示されてい
6:
                                    るため、一般のCコンパイラでは
           printf ("int a=%d\n",a); 初期化機能が働く.
7:
           printf ("static int b=%d\fm",b);
8:
9: }
```

〈コンパイル例〉

A>cc init1

BD Software C Compiler v1.50a (part I)

5: Missing semicolon

6: Undeclared identifier: static static static tables Cではサポート

6: Missing semicolon

A>

非常に簡単なプログラムですし、他のコンパイラでは大抵コンパイルすることができますが、上記のようにエラーがでてしまいます。このような場合は、ローカル変数、あるいは外部変数を用いて変数を定義し、プログラム先頭でその変数に初期値を設定するようにします。

リスト3-2-2 初期値の設定プログラム

```
1: #include <bdscio.h>
 2:
 3: main()
 4: {
 5:
 6:
7:
             a = 100;
             b = 200;
 8:
                                 (意味あいは異なるが)初期化が達成される.
             printf ("int a=%d\frac{\frac{1}{2}}{n}",a);
 9:
             printf ("int b=%d\fm",b);
10:
11: }
```

A>cc init2

BD Software C Compiler v1.50a (part I) 35K elbowroom

BD Software C Compiler v1.50 (part II) 32K to spare

A>clink init2

BD Software C Linker v1.50 Linkage complete 43K left over

A>init2

```
int a=100
int b=200
```

A>

この方法は実行時間がかかる、オブジェクトのサイズが大きくなるという 点でエレガントな方法ではありません。しかし、他のCコンパイラにかけて も問題ありませんし、ローカル変数に初期値を設定し忘れることも無くなり ます。

それより、問題は大きな配列などに初期値を設定したい場合です。リスト 3-2-3は 0 から 7 をデコードし、 $1\sim0$ x 80 を戻り値として返す関数の例ですが、スタティック変数の初期値を設定可能であれば、次のようにプログラムすることができます。

リスト3-2-3 配列を初期化するプログラム

```
1: #include (bdscio.h)
 2:
 3: main()
 4: {
 5:
             int i:
 6:
            for (i=0; i<8; i++)
 7:
 8:
                      printf ("%4d %04x\frac{1}{2}n", i, decode(i));
 9: }
10:
11:
12: decode(n)
13:
             int
                      n:
14: {
15:
             static int array [] = \{1, 2, 4, 8, 16, 32, 64, 128\};
16:
             return (array[n]);
17: }
```

A>cc init3

BD Software C Compiler v1.50a (part I)

```
15: Undeclared identifier: static ← static 変数はサポートされていない.
```

- 15: Need explicit dimension size ← 配列宣言では[]の中に必ず要素の数が必要.
- 16: Undeclared identifier: array 15行めのエラーで array が登録されなかったため、連鎖的に発生したエラー.

A>

この場合は高々8通りですから、switch文と case 文を用いてもプログラミングできますが、もっと多くなってくるとそうもいきません。そこで、プログラムの実行開始時に外部変数に値を書き込み、関数中でその配列を参照するような方法をとることもできます。その例をリスト3-2-4に示します。

リスト3-2-4 配列に外部変数を用いた例

```
1: #include <bdscio.h>
2:
          array[16]; 外部変数として配列を宣言
3: int
4:
5: main()
6: {
           int i:
7:
           setarray (); ← 配列に値を代入しておく
8:
9:
           for (i=0: i<8: i++)
10:
                   printf ("%4d %04x\n", i, decode(i));
11:
12: }
13:
14:
15: decode(n)
16:
           int
                  n:
17: {
          return (array[n]):
18:
19: }
20:
21:
                    ―配列に値を代入する関数
22: setarray()
23: {
            array[0]=1;
24:
            arrav[1]=2;
25:
            array[2]=4;
26:
27:
            array[3]=8;
            array[4]=16;
28:
```

```
29: array[5]=32;
30: array[6]=64;
31: array[7]=128;
32:}
```

A>cc init4

BD Software C Compiler v1.50a (part I) 35K elbowroom BD Software C Compiler v1.50 (part II) 32K to spare

A>clink init4

BD Software C Linker v1.50 Linkage complete 43K left over

A>init4

A>

実際 BDS C で本格的なプログラムを作成する場合、最大のネックになるのはこの初期値の設定で、場合によってはアセンブラを使った方が良い時もあります(アセンブラによるテーブルピックアップはプログラムとしては非常に簡単です)。

また、intなどの整数は、この方法でも十分ですが、文字列(文字配列)の 初期化はなかなか難しい問題です。マニュアルには strcpy 関数を用いるとあ りますが、文字列が多い場合には無駄が多いので、賛成しかねます。それよ りも、プログラムの構造自体に関わりますが、あらかじめポインタ配列を用 意しておき、最初にその配列に文字列の先頭アドレスをセットするようにしておくと文字列はプログラムの一部として埋め込まれるので効率的だと思います。ただし、この方法は文字列への書き込みができません(元の文字列より短ければ不可能ではありません)。

リスト3-2-5 文字列へのポインタ配列を作る例

```
1: #include <bdscio.h>
    2:
    3: chai *array[3]; ★ 文字列へのポインタ配列の宣言.
    4:
    5: main()
    6: {
    7:
               int
                      i:
    8:
               setarray (); ポインタ配列に文字列の
                                  先頭アドレスをセットする。
    9:
              for (i=0; i<3; i++)
                      printf ("string %d=%s\n",i,array[i]);
   10:
   11: }
   12:
   13: setarray()
   14: {
   15:
               array[0] = "Good morning.";
             array[1]="Good afternoon.";
   16:
               array[2] = "Good night.";
   17:
   18: }
A>cc init5
BD Software C Compiler v1.50a (part I)
 35K elbowroom
BD Software C Compiler v1.50 (part II)
 32K to spare
A>clink init5
BD Software C Linker v1.50
Linkage complete
43K left over
```

string 0=Good morning. string 1=Good afternoon. string 2=Good night.

A>

いずれにせよ、本来のC言語では悩む必要も無い部分なので、将来の機能拡張を望みたいところです。

3-3 プリプロセッサの機能

プリプロセッサとはプログラムをコンパイルする前に文字列の置き換えを 行なったり、条件コンパイルなどによるソースリストの削除を行なう機能で、 C言語を使いこなす上で、非常に重要なポイントの1つです。

BDS C, α-Cのプリプロセッサには次のようなものがあります.

#define

#unde f

#include

i f

#ifdef

#ifdef

#else

#endif

これらのうち、#define は文字列の置き換え、#include はファイルの取り込み、それ以降は条件コンパイルのために用いられます。上記以外のプリプロセッサはBDS Cでは削除されています。また、プリプロセッサではありませんが、コメントもコンパイラの重要な機能です。

(1) # define

#define は次のような書式で用いられます。

#define 定義名 定義値

一番多い使い方は次のように文字列に数値を設定する例です。

リスト3-3-1 数値展開の例

```
#include <bdscio.h>
#define MAXNUM 10
main()
        int i;
        for (i=0; i<MAXNUM; i++)
                 printf ("%d¥n",i);
}
A>define1
1
2
3 4 5 6
7
8
9
A>
```

定義名と定義値の間はスペースが区切りであるため、スペースを含む文字列を定義名として指定することはできません。また、この方法で指定した数値は数値をシンボル名で参照するわけではなく、

[&]quot;MAXNUM" → "10"

と別の文字列に置き換えるだけです。ここで MAXNUM は大文字になっていますが、 C 言語では定数は大文字で記述するのが一般的で、小文字でも別に動作には問題ありませんが、インデント(字下げ)と共に C 言語の標準プログラミングスタイルの1つです。

次に、 #define は、 単純な文字列置き換えだけでなく、 引数を持ったマクロ展開も行なうことができます。

リスト3-3-2 マクロ展開の例

A><u>define2</u> abcdefghijklmnopqrstuvwxyz ◆ 大文字が小文字に変換される。

A>

マクロ展開と言えども単なる文字列置き換えに過ぎないということには注意する必要があります。数値以外に関数呼出しや演算式を書く可能性がある部分には、引数自体を()で囲み、評価の順序などが誤らないようにする必要があります。リスト3-3-3は()を付けなかったため、誤った答となる例です。

リスト3-3-3 誤ったマクロ展開の例

```
#define minus(n) -n
main()
       printf ("%d", minus (10+2));
}
A>cc define3 -p
BD Software C Compiler v1.50a (part I)
   1:
  3: main()
   4: {
   5: printf ("%d",-10+2);
   6: }
                            minus(10+2) がこのように
  36K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
A>clink define3
BD Software C Linker v1.50
Linkage complete
  43K left over
A>define3
A>
```

コンパイル時の一p はプリプロセッサを通ったソースファイルがどのように変化したかを確認するために設けられているオプションで、例のように実際に展開された様子が画面に出力されます。この例では、

と展開されてしまいます。-(10+2) としたいわけですから、次のように修正しなければなりません。

リスト3-3-4 正しいマクロ展開の例

```
#define minus(n) - (n) ← nをカッコで囲んでいる点に注意.
main()
       printf ("%d", minus (10+2));
A>cc define4 -p
BD Software C Compiler v1.50a (part I)
  1:
  3: main()
  4: {
  5: printf ("%d", -(10+2));
  6: }
                              ·( ) つきで展開されている.
  36K elbowroom
BD Software C Compiler v1.50 (part II)
 32K to spare
A>clink define4
BD Software C Linker v1.50
b:Linkage complete
```

A><u>define4</u> -12 ← 答は正しい.

43K left over

慣れるまで時間がかかるとは思いますが、リスト 3-3-3のような例は見落 しがちなので、注意してください。

(2) # undef

#undefは #define による定義を抹消する場合に用います.

#define により同じ名前を定義すると後のものが優先されますが、入れ子になるわけではなく、#undefすると前の定義は残りません。

リスト 3-3-5 #undef の例

```
main()
#define DEF
             10
        printf ("DEF1=%d\fm", DEF);
#define DEF
                100
        printf ("DEF2=%d\fm", DEF);
       DEF
#undef
        printf ("DEF3=%d\fm", DEF);
}
A>CC UNDEF -P
BD Software C Compiler v1.50a (part I)
  1: main()
  2: {
  3:
  4: printf ("DEF1=%d\fm", 10);
                                         最後に定義された内容が
  5:
                                         有効になる。
        printf ("DEF2=%d\n", 100);
  6:
  7:
  8:
       printf ("DEF3=%d\n", DEF);"
                                          定義が抹消されているた
                                          めエラーとなる.
  9: }
  8: Undeclared identifier: DEF
```

このようなプログラムを作ることはあまりありませんが、知っておくと便利なこともあるでしょう。

(3) #include

#include はファイルの取り込みを指定しますほとんどの BDS C および α -C のプログラムでは、プログラム先頭に

#include <bdscio.h>

というヘッダーファイルの取り込みを記述します。この中にはいくつかの外部変数の宣言、構造体などの定義が記述されており、BDS Cで用意されている標準関数を使う場合には必ず指定する必要があります。

この bdscio.h は一般のC言語では stdio.h(Standard Input Output,標準入出力)という名前になっている場合が多いので、他のC言語に移植する場合には変更する必要があります。

次に, #include では, 書式に2種類あり,

#include <filename>
#include "filename"

の両者があります。ファイル名を<>で囲んだ場合にはログインディスク(プロンプトがA>の場合はドライブA)から捜され、""の場合はソースファイルと同じディスクから捜されます。実際のプログラム開発の場合、コンパイラ本体はドライブAに置いておくことが多いのでbdscio.hなど共通のヘッダーファイルも一緒にして<>で囲み、それ以外ユーザー側で作成したヘッダーファイルなどはソースファイルなどと一緒に異なるディスクに置き、""で囲むようにすると良いでしょう。

x なお、x 、""の双方ともファイル名にドライブ(x 、x B:x 。を記述するとそちらが優先されます。

目的のファイルが見つからないとそこでコンパイルは中断されます.

(4) 条件コンパイル

#if, #ifdef, #ifndef, #else, #endifはすべて条件コンパイルを指定するプリプロセッサですが、次のように用います。

#if 式 : #else : #endif

あるいは

#ifdef 識別子
:
#else
:
#endif

あるいは

#ifndef 識別子 : #else : #endif

if の場合は式の値が真(0 以外)であれば # else までをコンパイルし、それ以降 # endif までをスキップします。偽(0) なら # else 以降 # endif までをコンパイルします。 # ifdef、 # ifndef もほとんど同じです。 # ifdef では識別子 (名前)が # define によって定義されていれば、 # else までをコンパイルし、 # endif までをスキップします。 # ifndef は # ifdef の逆になります。

いずれの場合も #elseは省略することが可能です。また、これらの指定はネストすることができます。

リスト3-3-6 条件コンパイルネストの例

```
#define TEST1 1
main()
#if TEST1
        printf ("#if TEST1 compiled.\n");
 #ifdef TEST2 ← TEST 2 は定義されていない.
        printf ("#ifdef TEST2 compiled. Yn");
 #else
        printf ("TEST2 #else compiled. #n");
 #endif
#else
        printf ("TEST1 #else compiled. #n");
#endif
A>cc ifdef -p
BD Software C Compiler v1.50a (part I)
   1:
   3: main()
   4: {
   5:
       printf ("#if TEST1 compiled. #n");
   6:
                                                  この2つだけが
   7:
                                                 > 有効となる.
  10: printf ("TEST2 #else compiled. \n");
  11:
  15: }
  36K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
```

A>clink ifdef
BD Software C Linker v1.50
Linkage complete
43K left over

A>ifdef

#if TEST1 compiled.
TEST2 #else compiled.

A>

なお,この条件コンパイルはデバッグ時などデバッグ用プログラム部分を指定しておき,プログラム完成時に設定を変えるだけで削除したい場合に用います.

(5) コメント

コメントは文字列 /* から */ までをコメントとみなし、コンパイル時に 無視します。コメントはネストすることが可能で、

/ *

/* comment */

* /

という形式で使うこともできます。

3-4 浮動小数点演算

整数演算しかできない BDS Cにとって、小数点以下の値を含んだ数や、あるいは大きな数を扱う事のできる浮動小数点数演算パッケージ(float)は魅力あるものです。これらはうまく使うと非常に効果的にプログラムを作成す

ることができます.

しかし、関数の形でしか利用できませんから、互換性の点では他のCコンパイラにかける可能性がある場合には問題外となります。あくまで BDS Cで実数を使いたい方が用いるものと考えるべきでしょう。

しかし、数個の実数計算がある、あるいは平均点の結果だけ小数点表示を 行ないたいといった場合にはかなり救われることも事実で、恩恵は少ないと は言えません。

さて、float 関数を説明する前に、一言コメントしておきます。

float 関数にはオーバーフロー・アンダーフローを抑止する機能がありません. 一般には演算を行なって扱える数値の範囲を越えた場合, 計算を打ち切って, 扱える数のうち最もその値に近い値を返すものが多いのですが, float 関数の場合, 全くデタラメな値を返してきます. これは致命的なバグです.

一般には、非常に小さい値、あるいは大きな値を扱う可能性がある場合には、適当な時点で計算を打ち切り、オーバーフロー、アンダーフローが起らないような形でプログラミングする必要があります。ちなみに、扱える値は絶対値が約0.340282e39~0.293874e-38の範囲の値と0です。

これらのパッケージは BDS Cの開発者であるゾールマン氏の作成ではないことも、メンテナンスの点で問題を残しているようです。 α-C が float, long パッケージを含んでいないのもそんな所に原因があるのかも知れません.

(1) float 関数の使い方

float の数値は BDS C では5バイトの char 型変数配列で表されます。例えば、2つの実数の加算を行なう場合、1スト3-4-1のようにします。

リスト3-4-1 実数の加算と操作例

```
#include <bdscio.h>
main()
        char
               a[5];)
               b[5]; 実数は5バイトのchar型配列で表わされる.
        char
               c[5]:
        char
       atof (a, "1.23");
       itof (b, 1); b=1
fpadd (c, a, b); c=a+b
       printf ("%f+%f=%f",a,b,c);
}
A>cc ftest1
BD Software C Compiler v1.50a (part I)
  35K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
A>clink ftest1 -f float
BD Software C Linker v1.50
Linkage complete
  39K left over
A>ftest1
1.230000+1.000000=2.230000
A>
 ここで、
    atof
             文字列を実数に直す関数
    itof
             int型の数値を実数に直す関数
     fpadd
              実数の加算をする関数
```

です。BDS Cでは実数は5バイトの文字配列でしか表わすことができず、 じかに数値を記述したりすることはできません。定数であろうとも、必ず文 字配列を用意し、atof、itof などの関数を用いて値をセットする必要がありま す。また、リンク時には例のように、必ず

clink filename -f float

とし、リンクファイルの後に一fオプションを付加してください。これがないと正しいリンクを行なうことができません。特に、floatパッケージの中には float 版の printf 関数が含まれており、このようにしないと通常の printf 関数が誤ってリンクされてしまうことになります。この場合、例のように %fを用いて実数を画面に表示することができなくなりますので、注意してください。

なお、一般の整数では

c = a + b;

とでも記述すれば良いプログラムが実数を用いた場合には例のようにかなり 面倒な処理を行なわなくてはならなくなります。非常に良く間違うポイント ですので、十分注意してください。

なお、floatパッケージの関数の戻り値は演算結果を格納したバッファーの アドレスです。従って、次のような使い方は可能です。

リスト3-4-2 関数の戻り値をそのまま使う例

#include <bdscio.h>

main()

char a[5];
char b[5];
char c[5];

```
atof (a, "1.23");
itof (b,1);
printf ("%f+%f=%f", a, b, fpadd(c, a, b));
}
```

A>cc ftest2
BD Software C Compiler v1.50a (part I)
35K elbowroom
BD Software C Compiler v1.50 (part II)
32K to spare

A>clink ftest2 -f float
BD Software C Linker v1.50
Linkage complete
39K left over

A>f test2

1.230000+1.000000=2.230000

A >

このように、関数の戻り値をそのまま使うと、一般の演算の場合のように表現できるので、リストが短く、見やすくなります。また、floatパッケージの内部では実数演算用の専用のメモリを確保しており、いったんそこへ値をコピーしてから演算しますので、入力と出力に同じ文字列を指定することも可能です。(そのかわり、ROM化などの用途には使えません。)

リスト3-4-3 入力と出力に同じ文字列を指定した例

```
#include <bdscio.h>
main()
{
          char a[5];
          char b[5];
```

}

A>cc ftest3

BD Software C Compiler v1.50a (part I) 35K elbowroom BD Software C Compiler v1.50 (part II) 32K to spare

A>clink ftest3 -f float
BD Software C Linker v1.50
Linkage complete
39K left over

A>ftest3

1.230000+1.000000=2.230000

A>

ただし、ここで注意しなければならないことは、実数 a は fpadd(a, a, b) の 演算により a+b の内容に置き換えられてしまうことです。 つまり、

 $fpadd(a,a,b) \rightarrow a=a+b$

であることを念頭におくべきです。さもないと次のような誤りを犯すことになります。

リスト 3-4-4 fpadd の使う位置を誤った例

#include <bdscio.h>

main()
{

```
char a[5];
char b[5];

atof (a, "1.23");
itof (b, 1);
printf ("%f+%f=%f",a,b,fpadd(a,a,b));
```

A>cc ftest4
BD Software C Compiler v1.50a (part I)
35K elbowroom
BD Software C Compiler v1.50 (part II)
32K to spare

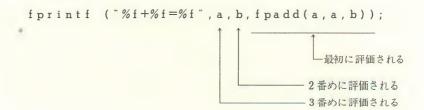
A>clink ftest4 -f float
BD Software C Linker v1.50
Linkage complete
39K left over

A>ftest4

}

2.230000+1.000000=2.230000 a の値はすでに2.23になっている.

BDS C, α -C では、関数への引数は最後から評価されます。



従って、この例では fpaddの関数が先に実行され、aの値がa+bに変化してしまってから最初に記述されている a を評価することになるため、例3-4

-4のようにおかしなことになるわけです。例 3-4-3 のように printf 文を分割するか、別の配列に出力しなければなりません。

なお、関数への引数が複数ある場合、どういう順序で評価するかはC言語 仕様上定まっていません。BDS Cでは最後から行ないますが、他のコンパ イラでは異なる順番で評価する場合があるため、このように引数の評価順序 に依存するプログラミングは好ましくありません。

(2) fplong. h

float 関数を使う場合、実数が5文字の文字列であるということはかなり大きな制約となります。プログラム中でも、

```
function()

char a[5];
```

という形で宣言すると、この文字配列 a は果して通常の文字列として用いるのか、実数として用いるのか判りにくくなります。そこで、次のような形で、実数が宣言できると便利です。

```
float a;
float k,1;
```

このため、実数を 5 文字の文字列から構成される構造体で表し、その構造体名を float とするヘッダーファイルを用意しました。このファイルでは同時にいくつかの関数についてマクロ定義を行なっているので実行がやや高速になる他、倍精度整数を用いる long パッケージの関数についても同様な定義を行なっていますので、

```
_long a;
_long k,l;
```

という形で倍精度整数が宣言できます。long は関数名と同じなので、使えないため、_long になっています。

リスト 3-4-5 fplong. h

```
float macro function
#define fpnorm(opl)
                                                                                   正規化
                                  fp (0, op1, op1)
#define fpadd(r,op1,op2)
                                  (fp(1, r, op1, op2), r)
                                                                                   加算
                                                           /* r <- op1+op2 */
                                  (fp(2, r, op1, op2), r)
                                                            /* r <- op1-op2 */
                                                                                   減算
#define fpsub(r, op1, op2)
#define fpmult(r,op1,op2)
                                  (fp(3, r, op1, op2), r)
                                                           /* r <- op1*op2 */
                                                                                   乗算
#define fpdiv(r, op1, op2)
                                  (fp(4, r, op1, op2), r)
                                                            /* r <- op1/op2 */
                                                                                   除算
#define ftoa (r.op)
                                  (fp(5,r,op),r)
                                                            /* float to ascii */ 実数→文字列变换
        long macro function
#define itol(r,op)
                                  long(0,r,op)
                                                            /* int to long */
#define lcomp(op1, op2)
                                  long (1, op1, op2)
                                                            /* if (op1==op2) 0
                                                                                   比較
                                                               op1>op2 ? 1:-1; */
#define ladd(r,op1,op2)
                                  long(2, r, op1, op2)
                                                            /* r <- op1+op2 */
#define lsub(r,op1,op2)
                                  long (3, r, op1, op2)
                                                            /* r <- op1-op2 */
                                                                                   減算
                                                            /* r <- op1*op2 */
/* r <- op1/op2 */
                                                                                   乗算
#define lmul(r,op1,op2)
                                  long (4, r, op1, op2)
#define ldiv(r,op1,op2)
                                  long (5, r, op1, op2)
                                                                                   減算
                                                            /* r <- op1%op2 */
#define lmod(r,opl,op2)
                                  long (6, r, op1, op2)
                                                                                   剰余
struct _float
         {
                 char
                         _fpdim[5];
#define float
                struct _float
struct
        _1 g
                         _lgdim[4];
                 char
        };
#define _long
                struct _lg
```

なお、このヘッダーファイルを用いた場合、引数については次のようにポインタで宣言します.

```
function (a)
float *a;
.
```

また、fplong.h を用いるもう一つのメリットは、この中で定義されている関数に限り、 α -C でも用いることができるということです。これは、float、long

関数の本体部分は標準関数として deff2. crlの中に格納されているためです。 このヘッダーファイルの使用例については以降のサンプルを参考にしてく ださい。

(3) 三角関数

sin 関数は非常に一般的に用いられる関数ですが、マクローリン展開

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$
$$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

を用いる事により、容易に関数値を求めることができます。ただ、このような式で近似値を求めるのはfloatパッケージにはかなり荷が重いようです。特に、項の値が小さい場合、アンダーフローが起りやすく、プログラミングにはかなり気を使います。

ここで紹介する sin 関数はある程度以上値が小さくなるとそこで計算を打ち切り、アンダーフローを防ぐようにしています。

なお、ここでは前述の fplong. hのヘッダーファイルを用いています。

pnt には求めた値を格納する 5バイトの文字配列の先頭アドレスを指定します。 \deg は角度ですが、ここで与える値は実際の角度に10を掛けたものです。 すなわち、この関数では 0.1 度単位で指定できるということになります。これはリストの28行めで設定されている TENの値を10から100に直せば、0.01 度単位で利用できる \sin 関数になりますが、整数は $-32768 \sim 32767$ の値しか

取れないので、-327.68度から327.67度までしか使えない \sin 関数となります。適当にこの値を変更してください。

リスト 3-4-7は10度単位に sin 関数値を表示するプログラムです。結果を見れば判るように単精度ですので余り精度は高くありません。なお、展開式の10項までで計算を打ち切るようになっていますが、演算結果がある値(CALCMAX)より小さくなると計算を打ち切ります。

リスト3-4-7 sin.cと実行例

```
#include <bdscio.h>
1:
      #include "fplong.h"
 2:
      #define CNTMAX 10
3:
      main()
 4:
 5:
 6:
               float
                        s, t:
7:
               int
                        i:
 8:
9:
               printf ("SIN
                                   VALUE: Yn");
10:
               printf ("angle sin\u00e4n");
11:
               for (i=0: i <= 1800: i+=100)
12:
                        sin (t,i);
13:
                        printf ("%3d %15.10f\n", i/10, t);
14:
15:
               }
16:
17:
      }
18:
      sin(r.data)
19:
20:
               float
                        *r;
21:
               int
                        data:
22:
       {
23:
                         i, isign;
               int
24:
                         s, x, b, e, f, g, xx, h, k, sign, xy;
               float
                        ONE, RD, TEN, CALCMAX, RDD;
25:
               float
26:
                                                   これらは変数だが実際には定数と
                                                   して扱っている。それを明示するために大文字で記述してある。
27:
      /* Set values
                        */
               itof (TEN, 10);
28:
               atof (RD, "0.3046e-3");
                                           /* RDD*RDD */
29:
               itof (ONE, 1);
30:
31:
               atof (CALCMAX, "1e-30");
32:
```

```
33:
              atof (RDD, "0.0174533"); /* 3.1415927/180 */
34:
35:
      /* Variables initialize */
36:
              isign = -1:
37:
              itof (sign, isign);
38:
              itof (x, data);
39:
              fpdiv (x, x, TEN);
40:
              fcopy (xy, x);
41:
42:
              fpmult(s,xy,RDD);
                                      /* s is result */
43:
              fcopy (k,s);
44:
45:
              itof(e,1);
                                       /* kaijo variable
                                                                 */
46:
              itof (g, 1);
                                       /* kaijo result
                                                                 */
47:
              itof (f, 1);
48:
49:
              fpmult (xx,x,x);
50:
              fpmult (xx, xx, RD); /* xx = x*x*RD
                                                                 */
51:
52:
      /* Calculation */
               for (i=2:i < CNTMAX: i++)
53:
54:
55:
                       fpadd (e, e, ONE);
                                               /* e=e+1
                                                                 */
56:
                       fcopy (g,e);
                                                /* g=e
                                                                 */
57:
                       fpadd (e, e, ONE);
                                                /* e=e+1
                                                                 */
58:
                       fpmult (g, g, e);
                                                /* g=g*e
                                                                 */
59:
60:
                       fpmult (h, xx, k);
                                                /* h=xx*k
                                                                 */
                       fpdiv (k, h, g);
61:
                                                /* k=h*g
                                                                 */
62:
63:
                       if (fpcomp (k, CALCMAX) == -1) break;
64:
65:
                       fpmult (f, k, sign);
66:
                       fpadd (s,s,f);
                                               /* s=s+f
                                                                 */
67:
                       isign = -isign;
68:
                       itof (sign, isign);
69:
               }
70:
71:
72:
               fcopy (r,s);
73:
74:
      }
75:
76:
      fcopy(t,o)
77:
              char
                       *t, *o:
```

```
78:
      {
79:
              int
                      i:
80:
              for (i=0; i<5; i++)
                                       *t++=*o++;
81:
      }
82:
〈実行例〉
A>sin
        VALUE:
SIN
angle
        sin
  0
       0.0000000000
 10
       0.1736483000
 20
       0.3420208000
 30
       0.5000017000
 40
       0.6427912000
 50
       0.7660509000
 60
       0.8660359000
 70
       0.9397083000
 80
       0.9848297000
 90
       1.0000290000
100
       0.9848453000
110
       0.9397389000
120
       0.8660807000
130
       0.7661086000
140
       0.6428597000
150
       0.5000794000
160
       0.3421047000
170
       0.1737365000
180
       0.0000897514
A>
```

リストの中で用いられている fcopy という関数は実数へ値を代入する関数です。

```
fcopy(a,b)
float *a,*b;
```

で, 実数 b を実数 a に代入します。

```
char *cos(pnt,deg)
  char *pnt;
  int deg;
```

sin 関数と使いかたは全く同じです。こちらも同じくマクローリン展開式,

$$\cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$
$$= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

を用いていますが、一般には、sin 関数、cos 関数のどちらかが判れば、

$$\sin x = \cos (x - 900)$$
$$\cos x = \sin (x + 900)$$

を利用すれば良いでしょう。 リスト 3-4-9 は \cos 関数値を10度単位で表示するプログラムです。

リスト 3-4-9 cos.cと実行例

```
1:
      #include <bdscio.h>
      #include "fplong.h"
 2:
 3:
      #define CNTMAX 10
 4:
 5:
      main()
 6:
      {
 7:
               float s;
 8:
               int
                       i:
 9:
               printf ("COS VALUE:\frac{\frac{1}{2}}{n}");
10:
11:
               printf ("angle cos\n");
               for (i=0; i<=1800; i+=100)
12:
13:
14:
                        cos (s, i);
```

```
15:
                       printf("%3d %15.10f\n", i/10, s);
16:
               }
17:
      }
18:
19:
     cos(r, data)
20:
               float *r;
21:
               int
                       data;
22:
      {
23:
               int
                       i, isign;
24:
              float s, x, b, e, f, g, xx, h, k, sign;
25:
              float ONE, RD, TEN, CALCMAX:
26:
     /* Set values */
27:
28:
             itof (TEN, 10);
29:
              atof (RD, "0.3046e-3"); /* RDD*RDD */
30:
              itof (ONE, 1);
31:
              atof (CALCMAX, "1e-30");
32:
33:
     /* Variables initialize */
34:
              isign = -1:
35:
              itof (sign, isign);
36:
              itof (x, data);
37:
              fpdiv (x, x, TEN);
38:
39:
             itof(s,1);
                                      /* s is result */
40:
              itof(e,0);
                                       /* kaijo variable
                                                                 */
41:
              itof (g, 1);
                                      /* kaijo result
                                                                 */
42:
              itof (f, 1);
43:
              itof (k, 1);
44:
45:
46:
             fpmult (xx,x,x);
47:
             fpmult (xx, xx, RD); /* xx = x*x*RD
                                                                 */
48:
49:
      /* Calculation */
50:
              for (i=2;i<CNTMAX;i++)
51:
52:
                       fpadd (e, e, ONE);
                                                /* e=e+1
                                                                 */
53:
                       fcopy (g,e);
                                                /* g=e
                                                                 */
54:
                       fpadd (e, e, ONE);
                                                /* e=e+1
                                                                 */
55:
                       fpmult (g, g, e);
                                                /* g=g*e
                                                                 */
56:
57:
                       fpmult (h, xx, k);
                                                /* h=xx*k
                                                                 */
58:
                       fpdiv (k, h, g);
                                                /* k=h*g
                                                                 */
59:
```

```
92
                       第3章 BDS C特有の問題点
                       if (fpcomp (k, CALCMAX) == -1) break;
60:
61:
62:
                       fpmult (f, k, sign);
                                                /* s=s+f
                                                                 */
63:
                       fpadd (s,s,f);
64:
                       isign = -isign;
                       itof (sign, isign);
65:
66:
               fcopy (r,s);
67:
68:
69:
70:
71:
      fcopy(t, o)
72:
               char
                       *t, *o;
73:
74:
               int i;
75:
              for (i=0: i<5: i++)
                                       *t++=*o++;
76:
      }
77:
〈実行例〉
A>cos
COS VALUE:
angle
          COS
  n
        1.0000000000
 10
        0.9848086000
 20
        0.9396960000
        0.8660328000
 30
        0.7660572000
 40
 50
        0.6428066000
        0.5000258000
 60
 70
        0.3420528000
 80
        0.1736874000
 90
        0.0000448172
       -0.1735991000
100
110
       -0.3419686000
120
       -0.4999482000
130
       -0.6427380000
140
       -0.7659996000
150
       -0.8659878000
160
       -0.9396652000
170
       -0.9847926000
180
       -0.9999997000
```

さて、sin、cos 関数があれば tan 関数は

$$\tan x = \frac{\sin x}{\cos x}$$

で求めるのが簡単ですが、 \sin 、 \cos 関数と異なり、 $\tan(90^\circ)$ は ∞ となるため、ある値以上では実数がとりうる最大の値を返すように工夫しなければなりません。

以上、三角関数のサンプルを紹介しましたが、述べたように、アンダーフロー、オーバーフローする場合があるために、完全なプログラムを作ることはできません。プログラムを変更した場合には誤った値を発生してしまう可能性があることに注意してください。また、これらの関数はアルゴリズムの問題もあり、精度は4桁程度で処理速度も遅くなっています。実際に利用される場合にはあらかじめ許容できるかどうか確認してください。

なお、最後に申し添えておきますが、信頼性の高いソフトウェアを作成する場合、実数演算を必要とするなら、BDS Cを用いるにはかなり工夫する必要があります。



第 **4** 章 CP/Mサンプルプログラム

BDS C, α-Cで作成されるプログラムで最も多いものはなんといって もCP/M用のユーティリティプログラムです。あれば便利といったものか ら、言語、高度なアプリケーションに至るまで、多くがBDS Cで作成さ れています。

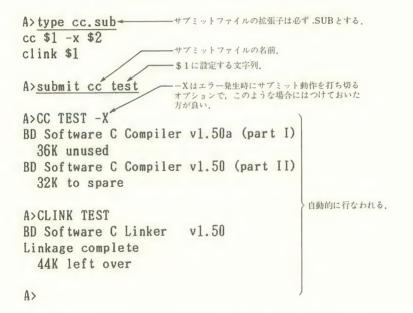
本章では、CP/Mに付属しているSUBMITを拡張したexsub(expanded submit) プログラムを紹介し、説明します。

4-1 拡張サブミットコマンド@.com

CP/Mにはバッチ処理用のコマンドとして、SUBMITが用意されており、 これにより、一連の作業を自動化することができます。

例えば、test. c というプログラムを開発する場合、あらかじめバッチ処理の順番を記述したファイル(.SUBファイル)を作っておくと、SUBMIT コマンドにより、自動的に comファイルまで作成することが可能になります。

リスト4-1-1 SUBMITファイルの例



また、SUBMITの実行を助ける XSUB というコマンドもサポートされており、プログラム中のキーボード入力を SUB ファイルから行なうことも可能になっています。このように、SUBMITは便利なコマンドですが、いくつかの欠点があり、使いづらい点があります。

そこで、より使いやすさを考慮した拡張 SUBMIT コマンドを作成しましたのでご紹介したいと思います。このプログラムはSUBMIT を包含していま

すので、今迄の SUB ファイルもそのまま利用することができます。

完成した exsub. com は @. com という名前にリネームしておくと、タイピングが楽になります。また、exsub をサポートするサブプログラム /. com を利用するとさらに有効に利用することが可能です。

exsub の特徴は以下のようなものです.

(1) 改行だけを入力することができる

例えば、PIPコマンドで複数のファイルを転送する際に、

A>PIP

*A := TEST.C

*A := B : PROG. ASM

* (改行のみ)

A>

などと、PIP コマンドを起動したあとで、キーボードから転送ファイル名を指定することができますが、SUBMIT コマンドでは XSUB を併用しても改行だけを記述することができず、PIP コマンドを終了させることができないため、必ず

PIP A := B : TEST.C

PIP A := B : PROG.ASM

として2回PIPコマンドを起動させる必要がありました.

(2) 間接的なサブミット呼出しが可能

SUBMIT ではその SUB ファイル中でさらに SUBMIT を実行すると SUB ファイルの残りの部分は無効になりました。exsub ではこれを改善し、あたかもサブルーチンのように別の SUB ファイルを実行させることができます。

(3) コントロールコードの入力が可能

^と大文字のA-Zを用いて、^Zなどとすればコントロールコードの入力が可能です。ただし、行先頭で^Cと記述してもリブートさせることはできません。

(4) 文字列置き換え可能

ファイル中の文字列 \$0~\$9をコマンドラインで指定した文字列に置き換えることができます。例えば、TEST.SUBというファイルに対し、

A>@ TEST RIE MITARAI

とすると,

\$ 0 → TEST

\$ 1 -> RIE

\$ 2 → MITARAI

に置き換わります。この機能はSUBMITと同じです。

(5) 特殊文字の入力が可能

\$\$→\$, **\$**^→^ という文字置き換え機能を持っています(SUBMITでは, **\$\$→\$**のみ).

なお、SUBMIT の補助コマンド、XSUB も@.com で利用できます。

〈SUBMIT の原理〉

このプログラムを理解するためには、SUBMIT がどのような原理で行なわれるかを知っておく必要があります。

まず, 例えば, サブミットファイルとして

pip a:=b:test.c
ren program.c=test.c

という内容を持つファイル(test.sub)があると、SUBMIT は\$1などファイル中の文字列の処理をしてから1行128バイトずつに展開し、最後の行から順に並べ換えたものを\$\$\$.SUBというファイルに書き込みます。

実際に実行してみると次のようになります

リスト4-1-2 SUBMITの実行

A>submit test

A>PIP A:=B:TEST.C

A>REN PROGRAM.C=TEST.C

pi動的に行なわれる.

SUBMITによって作成された\$\$\$.SUBファイルの中身は次のようになります.

リスト4-1-3 \$\$\$. SUBの構造

	── 最初の1バイトは文字数(16進, 20文字)															
	下線部が文字列となる。															
		/	/			/ 文字列の終了は 0, それ以降はゴミである。										
0100	14 52	45	4E	20	504	52	4F	47	52	41	4D	2E	43	3D	54	.REN PROGRAM.C=T
0110	45 53			43	00	24	6F	67	72	61	6D	2E	63	3D	74	EST. C. \$ogram.c=t
0120	65 73	74	2E	63	0D	0 A	1A	1A	1A	1 A	1A	1A	1 A	1A	1A	est.c
0130	1A 1/	1A	1 A	1A	1 A	1A	1A	1 A	1A	1 A	1A	1A	14	1 A	1A	• • • • • • • • • • • • • • • • •
0140 0150	1A 1A	111	1A	1A	1A	1A	1 A	14	1A	1 A	1A	1A	1A	1A	1A 1A	• • • • • • • • • • • • • • • • • • • •
0160	1A 1A	LI	1A	1A	1 A	1A	1A	1A	1A	1 A	1A	1 A	1A	1 A	1A	
0170	1A 1	1A	1A	1A	1 A	1A	1A	1 A	1A	1 A	1A	1 A	1A	1 A	1A	
0100	05 50	40		00	4.4	0.	0.0		0.0	- 1			- 1	0.0	4.0	
0180 0190	0F 50		50 2E	20	41	3A 24	3D 6F	67	3A 72	61	45 6D	53 2E	54 63	2E 3D	43	.PIP A:=B:TEST.C
0130 01A0	65 73		2E	63	0 D	0A	1A	1 A	1A	1 A	1A	1 A	1A	1 A	1A	.\$T.C.\$ogram.c=t
01B0	1A 1A		1A	1A	1A	1A	1A	1A	1A	1A	1A	1 A	1A	1A	1A	
01C0	1A 1A	2	1 A	1A	1 A	1A	1A	1 A	1A	1 A	1A	1 A	1 A	1 A	1 A	
01D0	1A 1A	4	1A	1A	1A	1A	1A	1A	1A	1A	1A	1 A	1A	1A	1A	• • • • • • • • • • • • • • • • • •
01E0 01F0	1A 1A	1A	1A	1A 1A	1A 1A	1A 1A	1A 1A	1 A	1A 1A	1A 1A	1A 1A	1 A	1A 1A	1A 1A	1A 1A	• • • • • • • • • • • • • • •
0110	In In	In	TU	T III	T II	In	Tu	TH	TH	In	T II	T II	1.11	TH	IM	

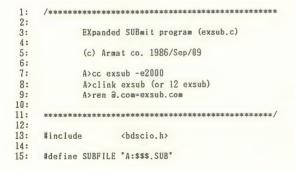
CP/MのシステムはこのドライブAの \$\$\$. SUBファイルの最後の1セクター (128バイト)を読み出してキーバッファーに格納すると同時に, \$\$\$. SUBファイルの最後の1セクターをデリートします。 \$\$\$. SUBファイルは SUBMIT ではデフォルトドライブ (B) の時はドライブB)に作成されるのですが, CP/Mシステム側はAドライブの\$\$\$. SUBしか検索しませんので, 実際にはログインディスクがAの場合 (A) のプロンプトが出ている状態)にしか利用することができません。exsubではどのような場合でも\$\$\$.SUBはドライブAに作成されるように改善されています。

リスト 4-1-3 のようなフォーマットでドライブ A 上に \$\$\$.SUB というファイルを作成すれば、それだけで CP/Mシステム側が自動的にプログラムの運転を行なってくれるわけです。なお、プログラム側でエラーなどの発生のため、サブミット動作を打ち切りたい時は \$\$\$.SUB というファイルを消去します。BDS Cのーx というオプションはエラー発生の際、サブミットを打ち切ってくれますが、実際には \$\$\$.SUB を消去しているだけです。

なお、\$\$\$.SUBファイルは通常の使い方ではディスク上に残ることはありませんが、リスト4-1-3のようなファイルの構造を知るには、SUBMITのバグを利用して、ドライブA以外に\$\$\$.SUBが作成されるようにするとデバッガーなどで簡単に内容を見ることができます。

リスト4-1-4に exsub.c のソースリストを示し、詳細に解説します.

リスト4-1-4 exsub.c



```
16: #define MAXLINE 200 ← もっと長い行数のサブミットファイルを
17: #define MAXCHAR 127 扱う場合にはここを変更する。
18:
19:
           sub file record struct */
20:
21:
    struct subsect 			 $$$.SUBの内部構造を表わす構造体タグ
22:
            {
23:
                    char
                           cnt;
                                        /* how many charactors
                                         /* string (null terminated)
                    char str[127];
24:
25:
            }:
26:
27:
    FILE iobuf:
                                         /* input file buffer
28:
29:
     struct subsect secbuf[MAXLINE];
                                         /* output buffer MAX=200 lines */
30:
     int
            linecnt:
                                          /* how many lines
31 :
32: main (argc, argv)
33:
            int
                   argc:
34:
            char *argv[];
35: {
36:
           char filename[20];
                                        /* filename buffer
                                                                      */
37:
           if (argc == 1)
38:
39:
                    printf ("Usage: @ submitfile arg1 arg2 ...");
40:
                  kexit ();
41:
           }
42:
43:
           strcpy (filename, argv[1]); /* set filename
                                                                      */
           strcat (filename, ".SUB");
44:
45:
46:
           if (ERROR == fopen (filename, iobuf))
47:
48:
                   printf ("Can't open %s", filename);
49:
                   kexit ();
50:
            submit (argc-1,&argv[1]); /* ex-sub main function
51:
            fclose (iobuf);
52:
53:
            writesub ();
                                         /* write $$$.sub
54: }
55:
56:
57:
    submit (ac, av)
58:
            int
                  ac:
59:
            char
                 *av[];
60:
    {
61:
                  buf[150];
                                         /* fgets line buffer
           char
                                         /* submit file buffer pointer */
62:
            char *pnt;
63:
64:
           linecnt = 0;
65:
           while (fgets(buf, iobuf))
66:
                   dellf (buf);
67:
68:
                   pnt = secbuf[linecnt].str;
69:
                   *pnt = 0:
70:
                   argexp (ac, av, buf, pnt);
                                               /* set $1,$2 ...
71:
                  if ((secbuf[linecnt].cnt = strlen(pnt))>MAXCHAR)
72:
73:
74:
                           printf ("Line overflow at line %d", linecnt+1);
75:
                          kexit ();
                   }
76:
```

```
linecnt++:
 77:
 78:
                                }
 79:
            }
 80:
 81:
           dellf(buf)
 82:
 83:
                                char *buf;
 84:
                                 while (*buf != '\n')
 85:
 86:
                                                  if (*buf == 0) break;
 87:
 88:
                                                  buf++:
 89:
 90:
                                *buf = 0:
 91: }
 92:
  93:
  94: argexp(ac, av, buf, pnt)
  95:
                                 int
                                                   ac:
 96:
                                 char
                                                  *av[]:
  97:
                                char
                                                  *buf,*pnt;
 98:
             {
 99:
                                 int
                                                  chars:
100:
                                 int
                                                  i:
101:
                                 int
                                                  strpnt;
102:
                               while (1)
103:
                                                    strpnt = index (buf, **); * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ * $ 
104:
105:
                                                    if (strpnt == ERROR)
106:
                                                    {
                                                                    107:
                                                                                                                                                           ままコピー
108:
                                                                     break:
109:
                                                    pnt = strcopy (pnt, strpnt, buf);
110:
                                                    buf += strpnt+1:
111:
                                                    if (isdigit(*buf))
112:
113:
                                                     {
114:
                                                                      i = *buf-0x30:
                                                                    if (i(ac) pnt = strcopy (pnt, strlen(av[i]), av[i]);
115:
116:
                                                                    buf++;
117:
                                                    else if (*buf == '$')
118:
119:
                                                     {
                                                                     *pnt++ = '$';
120:
121:
                                                                     buf++;
122:
                                                   }
                                                   else if (*buf == '^')
123:
124:
                                                    {
                                                                     *pnt++ = '^':
125:
126:
                                                                    buf++;
127:
128:
                                 }
129: }
130:
131:
132:
              strcopy(pnt, chars, buf)
                                                                                                       /* string copy with control charactors*/
133:
                                  char *pnt, *buf;
134:
                                  int
                                                    chars;
135:
136:
                                  char
137:
                                  while (chars--)
```

```
138:
              {
                      if (*buf == '^')
139:
                                            /* control charactor
140:
141:
                              buf++;
                              c = toupper(*buf++);
142:
                              if (c >= 0x40 || c < 0x60)
143:
                                                            *pnt++ = c-0x40:
                              if (chars--)
144:
                                           break:
145:
146:
                      else
                             *pnt++ = *buf++:
147:
148:
              *pnt = 0;
149:
              return (ont):
150:
      }
151:
152:
153:
      writesub()
                                             /* write submit file */
154:
155:
              struct subsect buf;
156:
              int
                     i:
157:
              int
                      fd:
158:
              if (ERROR != (fd = open (SUBFILE, 1))) seek (fd, 0, 2);
159:
              else if ( ERROR == (fd = creat (SUBFILE)))
160:
161:
                      printf ("Can't creat \"$$$.SUB\"");
162:
                      kexit ():
163:
              }
164:
165:
              for (i=linecnt-1: i>=0: i--)
166:
167:
                      if (1 != write (fd, secbuf[i], 1))
168:
169:
                             printf ("Disk full...");
170:
                             kexit ():
171:
172:
173:
              close (fd);
174:
175:
176:
177:
     kexit()
178:
      -{
179:
              180:
             exit ();
181:
      }
182:
```

[15~17行]

プログラム中で用いる定数およびファイル名を設定しています。ここで MAXLINE というのは変換可能な SUBファイルの行数を示しています。1行 につき、128バイト消費しますので200ラインで25Kバイト必要です。メモリ が許す範囲で値を大きく設定すればもっと多くの行数のファイルを処理できるようになります。

MAXCHAR はサブミットで可能な1行当りの文字数です.変更はできませ

h.

[21~25行]

subsect という構造体タグを定義しています。\$\$\$.SUBファイルの構造と 対応させ、扱いやすくしているわけです。

[29行]

subsect 構造をした構造体配列 secbuf を宣言しています。ここには実際にディスクに書き込む \$\$\$.SUB のイメージがそのまま作成されます。

[38~42行]

ここはプログラムの利用方法を示すUsage:を表示する部分です。@だけで改行すると、このメッセージが表示されます。

[43~50行]

ここで、目的の SUB ファイルの名前を文字列 filename に格納し、そのファイルを fopen でオープンします。 iobuf は27行めで宣言されていますが、バッファードファイル IO のためのバッファーのアドレスです。

もし、目的のファイルがオープンできなければ、エラーですからそこで中 断します。

[51行]

このプログラムの実際の処理をおこなう、submit という関数を呼び出します。

[52, 53行]

オープンしたファイルをクローズし、メモリ上のサブミットファイルのイメージを実際にディスクに書き込みます。

[57~79行]

実際の処理を行なう submit 関数です.

fgets により 1行ずつファイルを読み出し, \$ 1, \$ 2 などの処理を行ないながら,secbuf に書き込んでいきます.この時,一旦文字列 buf にその内容を書き込み,その長さがMAXCHAR(127) より短いことを確認しながら行なっていきます.

[82~91行]

関数 dellf です。fgets で取り込んだ1行は最後にYn コード (0 AH) が付いています。これを\$\$. SUB ファイルの中に書き込んではいけないので、削除する関数です。

[94~129行]

\$1,\$2などの文字列にコマンドラインからの文字列を展開する関数 argexpです。引数 ac, av は main 関数への引数とほとんど同じですが、51行めで判るように、あらかじめ不要な部分 (argv [0]) は除いてあります。pnt が書き込みのポインタ、buf が読み込みのポインタで、buf からの文字を読みながら、\$1,\$2などの処理を行ない、pnt に書き込みます。

[132~150行]

通常のstrcpyとほとんど同じ機能を持つ関数 strcopy です。cnt という引数により、文字列を指定の長さまでコピーします。また、^とA、^とZなど文字^と一緒のアルファベットを実際のコントロールコードに直します。

[153~174行]

関数 writesub は submit 関数によりメモリ secbuf 上に展開されている入力ファイルの内容をドライブAに \$\$\$.SUB というファイル名で書き込みます.この時、最後から書き込む点に注意してください.

なお、158、159行で、\$\$\$. SUB というファイルがオープンできれば(存在すれば)、そのファイルの最後から書き込むように

seek (fd,0,2);

とし、リードライトポインタをファイルの最後に設定します。また、もしオープンできなければ(存在しなければ)、ファイルを新たに creat しておきます。

これで、サブミット作業中にこのプログラムが呼ばれた場合、新たな作業内容が付け足され、あたかもサブルーチンのように実行されるわけです.

[177~181行]

エラーなどが発生した時には \$\$\$.SUB をデリートしてから終了します.

exsub.c はリストの7~9行にある要領でコンパイル・リンクし、最後に得られた COMファイルを @.com という名前にリネームしてください.

4-2 拡張サブミット補助コマンド/.com

@.comはSUBMITの機能をそのまま拡張したものですから、複雑な処理を行なわせることはできません。そこで、サブミット動作を補助するコマンドがあると便利です。

exsub をサポートするコマンド/(スラッシュ)の機能を説明します.

/.com はプログラム名 / の後にスペースを空けてサブコマンド名を記述することでサブミット実行時に色々な機能を利用することができるようになります。

/ if file 実行文字列

file が存在すれば実行文字列を実行します.

例えば、exsub. bak というファイルの有無をチェックし、exsub. c をコンパイルするかどうかを制御するなどといった事が可能です。

/ nif file 実行文字列

fileが存在しなければ、実行文字列を実行します。

/ eq 文字列 1 文字列 2 実行文字列

文字列1と文字列2が一致すれば実行文字列を実行します。

\$1, \$2などでコマンドラインから文字列を設定すれば, sub ファイルの 実行をコントロールすることができます.

/ neq 文字列 1 文字列 2 実行文字列

文字列1と文字列2が一致しなければ実行文字列を実行します。

/ skip 行数

無条件で行数分実行をキャンセルします.

このコマンドは / nif などと組み合せて用いられ、行数はサブミットファイル内でこの命令以降に記述されている行数が最大です。

〈例〉 / nif file.bak / skip 3

/ stop

サブミットファイルの実行を終了します. / skipと同様, / if などと組み合せて用いられます.

通常プログラム開発中には、エディターを用いてプログラムを書き直し、 さらにそのプログラムをコンパイルする必要があるわけですが、このとき、 バックアップファイルとして、拡張子が、bak というファイルが作成されます から、次のようなサブミットファイルを作ることができます。

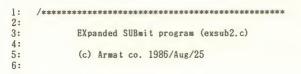
リスト4-2-1 /. com のサンプル

/ if exsub.bak @ cc exsub -e2000 ← もし exsub.bakが存在すれば、 @ cc exsub -e2000 を実行する.
/ if exsub2.bak @ cc exsub2 ← もし exsub2.bakが存在すれば、 @ cc exsub2 を実行する。 (注:このファイルを実行する時はリス) ト 4 − 1 − 1 の cc. subが必要。

これで、bakファイルがある場合(ファイルを更新した場合)には、コンパイルが行なわれ、無い場合にはコンパイルが行なわれず、無駄を省くことができるわけです。なお、この /. com は @. com ではなく、通常の SUBMIT と組み合せて使うこともできます。

それでは、/.com のソースプログラム、exsub 2.c について説明します.

リスト4-2-2 exsub 2.c



```
7:
             A>cc exsub2
             A>clink exsub2 (or 12 exsub2)
 8:
9:
             A>ren /.com=exsub2.com
10:
11:
     12:
13:
     #include
                    <bdscio.h>
14:
15:
            sub file record struct */
16:
17:
     struct subsect
18:
             {
19:
                                            /* how many charactors
                                                                            */
                     char
                             cnt;
20:
                             str[127];
                                           /* string (null terminated)
                                                                            */
                     char
21:
             };
22:
23:
     #define SUBCOM 1
                            /* SUBCOM is arg number in command line */
24:
     #define MAXCHAR 127
25:
     #define YES
                    - 1
26:
     #define NO
                     0
27:
    main (argc, argv)
28:
29:
             int
                     argc;
30:
             char
                     *argv[];
31:
     1
32:
             if (argc == 1)
33:
34:
                     printf ("Usage: / if testfile command arg... "n");
35:
                     printf ("
                                 / nif testfile command arg...\n');
36:
                                    / eq string1 string2 command arg... Yn*);
                     printf ("
37:
                     printf ("
                                    / neq string1 string2 command arg...\n");
38:
                     printf ("
                                    / skip lines n");
39:
                     printf ("
                                     / stop n");
40:
                     kexit ();
41:
             }
42:
43:
             if (!strcmp (argv[SUBCOM], "IF")) iffunc (argc, argv, YES);
44:
             else if (!strcmp (argv[SUBCOM], "NIF")) iffunc (argc, argv, NO);
45:
             else if (!strcmp (argv[SUBCOM], "EQ")) equal (argc, argv, YES);
46:
             else if (!strcmp (argv[SUBCOM], "NEQ")) equal (argc, argv, NO);
47:
48:
             else if (!strcmp (argv[SUBCOM], "SKIP")) skip (argc, argv);
             else if (!strcmp (argv[SUBCOM], "STOP")) kexit ();
49:
50:
             else
51:
              {
                     printf ("%s..?\n", argv[SUBCOM]);
52:
53:
                     kexit ();
54:
             }
55:
56:
57:
      iffunc (argc, argv, yorn)
58:
              int
                     argc;
59:
              char
                     *argv[];
60:
             char
                     yorn;
      1
61:
62:
             int
                    fd;
63:
             if (argc < SUBCOM+2) argerr ();
64:
```

```
fd = open (argv[SUBCOM+1],0);
 65:
              if ( (ERROR == fd && yorn == NO) !!
 66:
 67:
                   (ERROR != fd && yorn == YES) )
                      nextsub (argc-(SUBCOM+2), &argv[SUBCOM+2]);
 68:
 69:
              if (ERROR != fd)
                                close (fd);
 70:
     }
 71:
 72:
                                    /* write next command to submit file */
 73:
      nextsub(ct.args)
 74:
              int
                    ct;
 75:
              char
                     *args[];
 76:
      {
 77:
              struct subsect sec;
 78:
                      buf[200];
              char
 79:
              int
                      i;
 80:
              int
                      fd:
 81:
 82:
              strcpy (buf, args[0]);
 83:
              for (i=1; i<ct; i++)
 84:
 85:
                      strcat (buf, ");
 86:
                      strcat (buf, args[i]);
 87:
                      if (strlen (buf) > MAXCHAR)
 88:
 89:
                             printf ("Line overflow ..");
 90:
                             kexit ():
 91:
                      }
 92:
 93:
              sec.cnt = strlen (buf);
 94:
              strcpy (sec.str, buf);
 95:
                                             /* write disk file
                                                                 */
 96:
             if (ERROR == (fd = open ("A:$$$, SUB",1)))
 97:
 98:
                      printf ("Not in submit.");
 99:
                      kexit ();
100:
              101:
             if (1!= write (fd, sec, 1)) リードライトポインタをファイル の最後にセット.
102:
103:
104:
                     printf ("Disk full ..");
105:
                     kexit ();
106:
107:
             close (fd);
108:
     }
109:
110:
111:
      equal (argc, argv, yorn)
112:
             int
                   argc;
113:
              char
                     *argv[];
114:
              char
                     yorn;
115:
     {
             if (argc < SUBCOM+3)
                                   argerr ();
116:
             if (yorn == (!strcmp(argv[SUBCOM+1], argv[SUBCOM+2])))
117:
                     nextsub (argc-(SUBCOM+3),&argv[SUBCOM+3]);
118:
119:
     }
120:
121:
122:
     skip (argc, argv)
```

```
123:
              int
                    argc;
124:
              char *argv[];
125:
     {
                                                                              */
                                      /* how many skip lines
126:
              int
                      sline;
              if (argc < SUBCOM+1)
                                     argerr ();
127:
              sscanf (argv[SUBCOM+1], "%d", &sline);
128:
129:
              delline (sline);
130:
      }
131:
132:
                                                                              */
                              /* delete submit lines
133:
      delline(sline)
              int sline;
134:
135:
              struct subsect buf;
136:
137:
                      i:
              int
138:
              int
                      fd:
139:
              if (ERROR == (fd = open ("A:$$$.sub",1))) exit ();
140:
141:
142:
              buf.str[0] = 0;
               buf.cnt = 0:
143:
               for (i=1; i <= sline; i++)
144:
145:
                      if (ERROR == seek (fd,-i,2))
146:
147:
                               printf ("Can't delete lines");
148:
                               kexit ():
149:
150:
                       write (fd, buf, 1);
151:
152:
              close (fd);
153:
      }
154:
155:
156:
157:
       argerr()
158:
              printf ("Strange line..");
159:
160:
               kexit ();
161:
       }
162:
163:
      kexit()
164:
165:
               unlink ("A:$$$.SUB");
166:
167:
              exit ():
168:
      }
169:
```

[17~26行]

ここでは exsub.c と同じ構造体のテンプレートを宣言しています.

SUBCOMは if, eqなどのサブコマンドがコマンドライン中でどの位置にあるかを示す数値で、第1番めの引数ですから1となっています。

[33~42行]

/だけをタイプし、改行すると操作方法が表示されます。

[44~54行]

サブコマンドの文字列によって、その処理をする関数を呼び出します。新 しい機能を追加したければ、この部分に付け足せば良いでしょう。勿論、該 当するコマンドがなければエラーですから、エラーメッセージを表示して終 了します。

[57~70行]

サブコマンド if と nif を処理する関数です。yorn という引数は "yes or no" のことで、if (あれば) と nif (なければ)が両方同一の関数で処理できるように設けられています。この関数中で使われている open は実際のファイル読み書きのためではなく、ファイルがあるかないかを確認する目的で行なっています。ファイルの有無を確認するのは、他に wildexp パッケージなど用いてもできますが、こちらのほうが簡単です。

条件が合えば、関数 nextsub に必要な文字列の数とポインタ配列の先頭アドレスを送り、その内容を \$\$\$.SUB に書き込みます。

[73~108行]

文字列へのポインタ配列の先頭アドレスを args とし、ct個の文字列を \$\$. SUBファイルにアペンドする関数です。引数のフォーマットはコマンドラインからの引数を受け取る方法と全く同じです。

一旦 bufに文字列を設定し、それから実際に書き込む sec にコピーしています。これは展開する文字列をじかに sec に格納してしまうと文字列の長さが127文字を越えた場合に暴走する可能性があるためですが、実際にはコマンドラインそのものに127文字以上記述することができないため、不必要とも言えます。しかし将来プログラムを変更する際など、不要なトラブルに巻き

込まれないですみます。

[111~119行]

文字列の比較をして、もし条件が合えばコマンドラインの文字列を \$\$\$. SUBに書き込む eq および neg の処理ルーチンです。

[122~130行]

skip の処理をする関数です。sscanf で文字列を数値に変換し、デリートする行数を設定して関数 delline を呼び出します。

[133~154行]

サブミットで実行される処理を無効にする関数 delline です。無効にするためには \$\$\$.SUBファイルの最後から行数 *128 バイト分消去してしまえば良いのですが、BDS Cの標準関数ではファイルの一部を消去する関数がない(CP/Mの BDOSコールそのものに無い)ため、コマンド入力が無効になるよう文字数 0 となるようにファイルの内容を書き換え、実現しています(ディスクのディレクトリをじかに変更すれば可能です)。そのため、skipが実行された場合には skip する行数分、入力無しで実行が行なわれます。

[157~168行]

入力にエラーがあった場合に実行する関数 argerr, サブミット動作を中止する関数 kexit です。

なお、/の条件判断は実行された時点で行なわれます。@コマンド起動時に.bakファイルが存在していても/コマンド実行時までに消去されたりリネームされていれば、.bakファイルは存在しないと判断されます。

最後に、prg1.c、prg2.cという2つのBDS Cソースファイルとprgasm.csmというアセンブラソースファイルがある場合にそれをコンパイルする

SUBファイルとその実行例を紹介します.

リスト4-2-3 @と/コマンドの使用例

```
A>type prg.sub
```

```
/ if prgl.bak cc prgl -x → prgl.bakが存在すれば、prgl.cをコンパイル。
/ if prg2.bak cc prg2 -x → prg2.bakが存在すれば、prg2.cをコンパイル。
/ if prgasm.bak ② zcmr prgasm → prgasm.bakが存在すれば、すらにzcmrというサブミット処理を行なう。
era *.bak
clink prgl prg2 prgasm
```

A>type zcmr.sub

```
zcasm $1
mrasm $1.aaz ; (drive b)->.bbz
era $1.asz
load $1
era $1.hex
era $1.crl
ren $1.crl=$1.com
; $1.crl is ready
```

```
A>dir prg*.c*
A: PRG1 C : PRG2 C : PRGASM CSM ← Y-スファイル
```

A>dir prg*.bak

A: PRG1 BAK: PRG2 BAK: PRGASM BAK すべてバックアップ ファイルが存在している.

A>type prgl.c

```
A>type prg2.c
                                                               prg 2. cの内容.
#include <bdscio.h>
prg2(i)
          char
                   i:
          printf ("original=%d, decode=%d\fm", i, prgasm(i));
A>type prgasm.csm
function prgasm
                    hl
                              : return address
          pop
                    de
          pop
                              ; argument
          push
                    de
          push
                    hl
                                                   prgasm.csmの内容0~7
                                                   の値をデコードし、
                                                     0 \rightarrow 1
          1d
                    hl, table
                                                     1 \rightarrow 2
                                                     2 \rightarrow 4
          add
                    hl, de
                                                     3 \rightarrow 8
                                                     4 \rightarrow 16
          1d
                    1, (h1)
                                                     5 → 32
         1d
                    h, 0
                                                     6 \rightarrow 64
                                                     7 \rightarrow 128
         ret
                                                   に変換する関数。
: decode table
table: db
                    1, 2, 4, 8, 16, 32, 64, 128
endfunc
A>@ prg-
                  -サブミットの起動。
                   以下は自動的に行なわれる.
A>/ if prgl.bak cc prgl -x
A>CC PRG1 -X
                                               prgl.bakが存在するので、
BD Software C Compiler v1.50a (part I)
                                                prgl.cがコンパイルされた.
  35K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
```

A>/ if prg2.bak cc prg2 -x

A>CC PRG2 -X BD Software C Compiler v1.50a (part I) | prg2.cがコンパイルされた. 35K elbowroom

BD Software C Compiler v1.50 (part II) 32K to spare

prg2.bakが存在するので されない)

A>/ if prgasm.bak @ zcmr prgasm

prgasm.bak が存在するので zcmr というサブミットフ アイルが起動される.

A>@ ZCMR PRGASM

A>zcasm PRGASM Armat Z80-CRL preprocessor v1.0 Pass 1: End

Pass 2:

-Processing the PRGASM function.. PRGASM. ASZ is ready to be assembled.

A>mrasm PRGASM.aaz ; (drive b) -> .bbz Armat Z80 Assembler - VER 1.3

(c) Armat co. 1985, 1986 All Rights Reserved.

No Fatal error(s)

A>era PRGASM.asz A>load PRGASM

FIRST ADDRESS 0100 LAST ADDRESS 031F BYTES READ 002BRECORDS WRITTEN 05

A>era PRGASM.hex A>era PRGASM.crl NO FILE A>ren PRGASM.crl=PRGASM.com A>; PRGASM.crl is ready

zcmr内の処理.

```
A>era *.bak
```

A>clink prg1 prg2 prgasm
BD Software C Linker v1.50
Linkage complete
43K left over

zcmrの処理が終了すると 元のprg.subの処理に戻る。

〈サブミット動作終了〉

```
A>prgl
```

original=0, decode=1 original=1, decode=2 original=2, decode=4 original=3, decode=8 original=4, decode=16 original=5, decode=32 original=6, decode=64 original=7, decode=128

プログラムの実行

A>

400

Secretary to the second

G* 51 080

1 - 100 - 101 - 1

timitatan 1 Tana 1 gantan - I

of manage of the second of the

051 About 1

.

第 5 章

日本語処理

BDS C, α -Cでは基本的に日本語の処理ができるようにはなっていません。本来、これらは米国で開発されたものですから、文字としては英数字だけが想定されており、文字列処理用の標準関数などもそのままでは使うことができない場合がほとんどです。

しかし、C言語の基本的な仕様は(日本語、英語などといった)言語に左右されてはおらず、新しい関数を作成すれば本格的な日本語処理を行なうことが可能です。本章では日本語を使う際のポイントを解説し、ならびに実用的な日本語処理関数パッケージを紹介します。

5-1 BDS C, α-Cの変更

BDS C, α-Cでは基本的な文字として英数字だけが考えられていますので、カナ、漢字などはソースリストに記述してもコンパイラにかけた時点でアスキーコードのMSBが落とされてしまい、英数字に変化してしまいます。

リスト5-1-1 BDS Cでカナを用いた場合

```
A>type kana.c
main()
        printf ("מלים אול" וול BDS C ה"וו שמבילע."):
A>cc kana -p
BD Software C Compiler v1.50a (part I)
                                                         MSB(最上位ビ
                                                         ット)が落とさ
                                                         れるため,こ
   1: main()
                                                         ようになってしまう.
   2: {
   3: printf ("6EJ 5X<^EY I BDS C C^J B640>].
   4: }
  36K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
A>clink kana
BD Software C Linker v1.50
Linkage complete
  43K left over
A>kana
6EJ 5X<^EY I BDS C C<sup>↑</sup>J B640>]. 表示はもちろんこのようになる.
A>
```

リスト5-1-1の状況では、全く日本語の処理などできなくなります。し

かし、strcpy、strlen などの標準的な文字列処理関数はこの制限を持っているわけではなく、終端文字の0さえ確実にセットしてあればMSBは影響を受けません。そこで、コンパイル時にカナ、漢字のMSBがキャンセルされないようにすれば、何とか日本語処理ができるようになります。

この方法として次のようなものが考えられます。

- (1) コンパイラ自体を変更し、MSBがキャンセルされないように改造する.
- (2) ソースファイル用のプリプロセッサを作成し、カナ、漢字をコンパイラで正しく認識されるような形式に変更する。

(1)は簡単なパッチ処理で済みますが、本質的な方法ではないため、多くの制限が残ります。(2)は処理は完全ですが、別のプリプロセッサにソースファイルを通さねばならないため、コンパイル時の手間がかなりかかります。

実用的には両者を併用するのが便利でしょう。なお、本項で扱う漢字は、シフトJISコード方式のものです。

BDS Cはコンパイラ本体はcc. comおよび, cc 2. comという2つのプロググラムから成り立っていますが, このうち, 文字列の解析などを行なっているのはプリプロセッサ部であるcc. comです。cc. comの次の番地の内容を7FHからFFHに変更することにより, MSBのキャンセルを防ぐことができます。

BDS C v1. 50a 27C 7 H番地 α-C v1. 51 27F 1 H番地

操作例 5-1-2 cc. comの変更

A>ddt cc.com DDT VERS 2.2 NEXT PC 3A00 0100

```
-s27c7 ←
             — DDT などによって 27 C7 H 番地の内容を
              変更する(α-Cの場合は27F1H番地).
27C7 7F ff
27C8 22 .
-^C
                        -SAVEコマンドにより書き換えた内容を記録する.
A>save 57 cc.com
A>cc kana -p ← 変更を加えた BDS Cでテスト.
BD Software C Compiler v1.50a (part I)
   1: main()
   2: {
       printf ("カナハ オリシ"ナル ノ BDS C テ"ハ ツカエマセン.");
   4: }
  36K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
A>clink kana
BD Software C Linker v1.50
Linkage complete
  43K left over
A>kana
カナハ オリシッナル ノ BDS C デッハ ツカエマセン。◆ 正しく表示される.
```

この方法は非常に手軽ですが、残念ながら、次のような欠点を持っています。

(1) マクロ定義された文字列については有効でない。

プログラム内部で

A>

printf ("漢字テスト");

などと記述する場合には問題ありませんが、次のような場合は正しくコンパイルされません。

リスト5-1-3 マクロ定義でうまくいかない例

```
A>type define.c
```

A>cc define -p

BD Software C Compiler v1.50a (part I)

```
1:
3: main()
4: {
5: printf (*6);
6: }
```

5: String too long (or missing quote) ← この場合はコンパイルそのものができなかった。

A>

余り変更しないメッセージはじかにプログラム内部に置く場合が多いのですが、変更が考えられるメッセージなどはあらかじめ#defineを用いてマクロ 定義とする方がはるかに便利ですし、間違いも少なくなります。これはかなり大きな制約であると言えます。

(2) 特別な漢字を用いた場合に文字化けが発生する。

C言語では、文字¥ (JISコード、アスキーコードでは/)を文字、文字列中の特別な記号として用います。所が、この記号は5CHというコードを持っていますから、シフトJIS方式の2バイトコードで漢字を表示すると、2バイ

3: 4: }

トめがこのコードと一致することがあります。この場合、文字列の内容を誤って解釈してしまいますから、完全に文字が変化することになります。さらに具合が悪いことに文字列の最後にこの文字がある場合には文字化けでは済まず。コンパイルそのものができなくなります。

リスト5-1-4 文字化けする場合

```
A>cc kanjerr -p
BD Software C Compiler v1.50a (part I)
  1: main()
  2: {
  3: printf ("すべての漢字を正しく表現できません");
  4: }
                                      コンパイルでは正しいように
                                     見えるが……
 36K elbowroom
BD Software C Compiler v1.50 (part II)
 32K to spare
A>clink kanjerr
BD Software C Linker v1.50
Linkage complete
 43K left over
A>kanjerr
                                     -文字化けした文字列。漢字コード
すべての漢字を正しく阜りできません
                                     の2バイトめが5CHの文字はこのよ
                                      うになってしまう.
A>
              リスト5-1-5 コンパイルができない場合
A>cc f:kanjerr2 -p
BD Software C Compiler v1.50a (part I)
  1: main()
   2: {
```

3: String too long (or missing quote)

A>

パッチによる方法は非常に簡単で効果も大きいのですが、そこに上記のような問題が残されている事にも注意しなければなりません。

5-2 プリプロセッサを用いる方法

あらかじめソースプログラムそのものを変更してしまおうというのがプリプロセッサを用いる方法ですが、C言語ではもともと8ビットフルに用いた文字列を使ってはならないわけではなく、ある特定の条件のもとでは使用することが可能です。

"¥377¥376"

という文字列は特殊文字¥を用いて、0FFH、0FEHのコードを文字列に設定する例です。¥マークのあと8進3桁で数値を指定することにより0から0FFHまですべてのコードを文字列の中に埋め込めます。この方法を用いて、

"表現" → "¥225¥134¥214¥273"

という形に変換してしまうわけです。全く何を表現しているのか判らなくなりますが、すべての制約はなくなります。

それでは、この処理を行なうプリプロセッサktocをご紹介します。

リスト5-2-1 ktoc.c



```
9:
                 A>clink ktoc
10:
    ***************
11:
     #define DEBUG 0
12:
13:
     #include (bdscio.h)
14:
15:
          lbuffer[512]; /* get line buffer
16:
     char
17:
    FILE
          fpin;
                         /* input file buffer
                         /* output file buffer */
18:
    FILE fpout;
19:
                        /* 行カウンター
20: int
           lcnt;
21: char kaniif:
                        /* 漢字あり:1,カナのみ:0*/
22:
23:
    main (argc, argv)
24:
25:
     int
           argc:
26: char
           *argv[];
27: {
28:
            char
                  infile[20];
           char outfile[20];
29:
30:
31:
           if (argc < 2)
32:
                   printf ("書式: ktoc filename [-nk]\n");
33:
                  printf (" -nk: 漢字を使用しない場合");
34:
35:
                  exit ();
36:
37:
            strcpy (infile, argv[1]);
            strcpy (outfile, argv[1]);
38:
           strcat (infile, ".C");
strcat (outfile, ".CK");
39:
40:
41:
            if (strcmp (argv[2], "-NK"))
                                      kanjif = TRUE;
42:
                                      kanjif = FALSE;
43:
           else
44:
45:
            if (ERROR == fopen (infile,fpin))
46:
                   printf ("(%s)がオープンできません",infile);
47:
48:
                  exit ();
49:
            }
50:
            if (ERROR == fcreat (outfile, fpout))
51:
                   printf ("<%s>を作成することができません",outfile);
52:
53:
                   exit ():
54:
55:
            ktoc ():
                             /* メインルーチン */
            fputc (CPMEOF);
56:
57:
58:
           if (ERROR == fclose (fpout))
                  printf ("<%s>を正しくクローズできません",outfile);
59:
            else printf ("<%s>を作成しました",outfile);
60:
61: }
62:
63:
    ktoc()
64:
65: {
                   c1,c2;
66:
            char
67:
            char
                  *cpnt;
                               /* コメントのネストカウンター */
                   coment;
68:
            int
```

```
69:
 70:
              lcnt = 1;
 71:
              coment = 0;
 72:
              while (fgets (lbuffer, fpin))
 73:
 74:
       #if DEBUG
                      printf ("(%d)%s", coment, lbuffer);
 75:
 76:
       #endif
 77:
                      cpnt = lbuffer;
 78:
                      c1 = 0;
 79:
                      while (c2 = *cpnt)
 80:
                       {
 81:
                              fputc (c2);
 82:
                              cpnt++;
 83:
                              if (c1=='/' && c2=='*') comcnt++;
                              else if (c1 == '*' && c2 == '/')
 84:
 85:
 86:
                                      if (--comcnt < 0)
 87:
 88:
                                              printf ("%d: %s", lcnt, lbuffer);
 89:
                                             printf ("コメントが正しくありません");
 90:
                                             exit ();
 91:
                                      }
 92:
 93:
                              if (!comcnt)
 94:
 95:
                                      if (c2 == '\f'') cpnt=skipchar (cpnt);
                                      if (c2 == '"') cpnt=setstr (cpnt);
 96:
97:
98:
                              c1=c2;
99:
100:
                      lcnt++;
101:
102:
              if (coment !=0)
103:
                      printf ("コメントが閉じていません");
104:
105:
                      exit ();
106:
107:
     }
108:
109:
110:
     fputc(c)
111:
              char c;
112:
              if (c=='\n')
                             putc ('Yr', fpout);
113:
114:
              if (ERROR == putc (c, fpout))
115:
116:
                      printf ("ディスクが一杯です");
117:
118:
                      exit ();
119:
              }
      #if DEBUG
120:
121:
              putchar (c);
122:
      #endif
123:
      }
124:
125:
126:
      skipchar (pnt)
127:
             char
                    *pnt;
128:
```

```
129:
               while (*pnt != '\'')
130:
131:
                        fputc (*pnt);
                        if (*pnt == '\f') fputc (*++pnt);
132:
133:
                        else if (*pnt == '\0')
134:
135:
                                printf ("%d: %s", lcnt, lbuffer);
                                printf ("文字が正しく記述されていません");
136:
137:
                                exit ():
138:
139:
                       pnt++;
140:
141:
               fputc (*pnt++);
142:
               return (pnt);
143:
       }
144:
145:
146:
       setstr(pnt)
147:
               char
                       *pnt;
148:
149:
               char
                       kflag:
                                        /* 漢字 1バイトめのフラグ */
150:
               kflag = FALSE;
151:
152:
               while (1)
153:
154:
                        if (kflag)
155:
                        {
156:
                                if (*pnt=='\\\' \\ +pnt>=0x80)
157:
158:
       #if DERUG
159:
                                        printf ("(2nd) \\%03o", *pnt);
160:
       #endif
161:
                                        fprintf (fpout, "\\1030", *pnt++);
162:
163:
                                else
                                        fputc (*pnt++);
164:
                               kflag = FALSE:
165:
166:
                        else if (*pnt==' "")
167:
168:
                                fputc (*pnt++);
169:
                              fputc (*pnt++):
170:
171:
                       else if (*pnt>=0x80)
172:
       #if DEBUG
173:
                                printf ("(1st) \\%03o", *pnt);
174:
175:
       #endif
                                fprintf (fpout, "\\030", *pnt);
176:
177:
                                if (kaniif
178:
                                    ((*pnt>=0x80 && *pnt<0xa0) ||
179:
                                     (*pnt>=0xe0 && *pnt<0xfd)) )
180:
                                        kflag = TRUE;
181:
                                pnt++;
182:
183:
                       else if (*pnt == '"')
184:
                        {
185:
                                fputc (*pnt++);
186:
                                return (pnt):
187:
188:
                       else if (*pnt == '\0')
```

```
189:
                     {
                             printf ("%d: %s", lcnt, lbuffer);
190:
                             printf ("文字列が長過ぎるか¥"マークが落ちています");
191:
192:
                             exit ();
193:
194:
                     else
                            fputc (*pnt++);
195:
196:
     }
197:
```

〈変数および関数表〉

このktocはシフトJIS漢字コード方式の場合、その第1バイトめが $80H\sim9$ FHおよび $E0H\sim$ FCHにあることを利用しています。

プログラム内部の文字列をすべてチェックし、文字列の中に80 H以上のキャラクタがあればそれを¥+8進3桁の形式に変換して出力します。ただし、漢字の2バイトめの場合、そのコードが¥そのもののコード(5 CH)と同じだとコンパイル時に誤って変換されてしまいますので、5 CH、および80 H以上の場合のみ¥+8進3桁の形式に直します。それ以外は文字キャラクタとなりますが、これはBDS Cでは文字列として最大255文字までしか記述できず、¥+8進3桁で記述した場合、見掛け上文字列が長くなり、コンパイル不能になることがあるからで、少しでも短くなるようにしてあります。

なお、漢字を用いず、カタカナだけを利用する場合は、逆に80H~9FH、E0H~FCHに設定されているグラフィックキャラクタを利用すると問題を生ずるため、ktocプリプロセッサにかける時にオプションを指定できるようになっています。

ktoc の使い方は次のとおりです。

書式: ktoc filename [-nk]

filenameに拡張子"・c"はいりません。また出力ファイル名はfilenameに拡張子".ck"を付けたものとなります。またオプションの-nkは漢字を使わない場合に指定します(no kanji)

正しく変換されると、"〈filename.ck〉を作成しました"というコメントが 出て終了しますので、あとは

cc filename.ck

という通常の方法でコンパイルすることができます。注意することはコンパイル時に必ずファイル名に ".ck" という拡張子を忘れないことで、これを忘れると元のファイルがコンパイルされることになります。

なお、このktocはエラーメッセージなどがすべて漢字で記述されているため、5-1 項で示された変更を行なったBDS Cあるいは α -Cでなければコンパイルできません(メッセージを英字にしておけばコンパイルできます)。しかし、ktocで変換されたファイルは cc.com 変更前のものでもコンパイルできますので、このプログラムを入力された方は、ktoc.c そのものをktocにより変換し、正しく動作するか確かめると良いでしょう。なお、ktocにかけるプログラムでは文字列を1行以内で記述してください。文字列中で改行するとエラーとなります。また、文字(''で囲んだもの)中の漢字などは変換されません。

次に操作例を示しておきますので参考にしてください。

操作例 5-2-2 ktocの操作方法

```
A>cc ktoc -e2100←—ktocのコンパイルはカナ対応に変更したBDS C, あるいはα-Cで行なう.
BD Software C Compiler v1.50a (part I)
  32K elbowroom
BD Software C Compiler v1.50 (part II)
  29K to spare
A>clink ktoc
BD Software C Linker v1.50
Linkage complete
  38K left over
A>cc ktest1 -p
BD Software C Compiler v1.50a (part I)
  1: main()
  2: {
  3: printf ("表現力¥n"); 
4: printf ("十人¥n"); ー見正しくコンバイルされるかのように見えるが.
   5: }
  36K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
A>clink ktest1
BD Software C Linker v1.50
Linkage complete
  43K left over
A>ktest1
阜り力)
            ――実行すると文字化けしてしまう.
```

```
A>ktoc ktest1 ← 上記のktest1.cをktocに通してみる.
<F:KTEST1.CK>を作成しました
A>type ktest1.ck
main()
       printf ("¥225¥134¥214¥273¥227¥315¥n"); 漢字を含まない
       printf ("\217\134\2201\n");
}
A>cc ktest1.ck
BD Software C Compiler v1.50a (part I)
  36K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
A>clink ktestl
BD Software C Linker v1.50
Linkage complete
43K left over
A>ktest1
表現力
       正しく表示される.
A>type ktest2.c
#define MSG1
               "マクロ定義も使えます、¥n"
#define MSG2
               "カタカナ モ 併用 デ"キマス¥n"
main()
       printf (MSG1);
       printf (MSG2);
       printf ("表現力¥n");
       printf ("十人\n");
}
A>cc ktest2 -p
BD Software C Compiler v1.50a (part I)
```

```
1:
  4: main()
  5: {
  6: printf (");
  7: printf (MSG2);
  8: printf ("表現力\n");
  9: printf ("十人\n");
 10: }
  6: String too long (or missing quote) ← ktest2.clt
                                           コンパイルすら不能.
A>ktoc ktest2 ktocに通してからコンパイル.
<F:KTEST2.CK>を作成しました
A>cc ktest2.ck
BD Software C Compiler v1.50a (part I)
 35K elbowroom
BD Software C Compiler v1.50 (part II)
 32K to spare
A>clink ktest2
BD Software C Linker v1.50
Linkage complete
 43K left over
A>ktest2
マクロ定義も使えます.
カタカナ モ 併用 デ"キマス
                    制限なしに
                     すべての表示が可能.
表現力
十人
A>
```

5-3 日本語処理パッケージ

さて、前々項、前項で、BDS C、 α -Cにより、日本語処理を行なう用意は整いました。

しかし、漢字などのシフトJISコードは基本的に漢数字の8ビットのコード体系とコンパチビリティを持たせた形にしてあるものの。C言語では日本語が標準ではないために、日本語文字列の処理を行なう関数のすべては有効に使うことができません。

そのため、BDS C、α-C用に専用の日本語処理パッケージを作成しました。この関数パッケージを用いることで、日本語文字列の処理が簡単に行なうことができるようになります。是非、有効に使っていただきたいと思います。まず、この日本語処理パッケージに含まれる関数について説明します。

kgetline(buf,cnt)
char *buf;
int cnt;

bufに内容を格納したいバッファーのアドレス, cntに入力可能な最大文字数+1をセットするとコンソールから1行入力し, その結果をbutから格納します。戻り値は入力した文字数になります。ただし, リターンキーは文字列および文字数には含まれません。

この関数で注意することは1バイト系の文字を入力している場合には最大 cnt-2 文字までしか入力が行なわれないことです(終端文字の0を除く).もし、cnt-2 文字めが漢字など2バイト系文字の1バイトめであった場合には、2バイトめもコンソールから入力してバッファに格納します。

kgetline (buf, 11) の場合

カナ, 英数字のみ : 9 文字まで

第9文字が漢字の1バイトめ:10文字まで入力

getline, gets などは文字列を入力する場合の基本的な関数ですが、漢字などを入力する場合にはうまくいかない場合があります。それは2バイト系の

文字が存在するためにgetlineの文字数制限により、無情にも第1バイトのみで入力が打ち切られてしまう場合があることと、BS、DELなどのキーを併用した際に漢字の第2バイトだけしか消去されず、2回キーを押さないと正しく文字を消去することができないことです。特にDEL、BSなどのキーについての問題は、通信などでも面倒な部分です。

kgetlineではDELキー (18H), バックスペース (7FH) で前の1文字を削除する処理を行ないます。漢字などの場合は,正しく2バイト分消去します。また。コントロールXによる入力行のキャンセルもサポートしています。なお,これらの処理ではバックスペース (Yb) およびカーソルキャラクタを用いてBDOSコールでカーソル移動を行なっていますので、CP/MのBIOSの作成法およびBDS Cのバージョンによって若干動作が異なる可能性があります。また、TABキー(コントロールI,09H)の入力があった場合はDEL、コントロールXなどのキーが正しく動作しません。注意してください。

【注意】 TABの問題についてはBDOS コールのファンクション1 (1文字 入力), 2(1文字出力), 9(文字列出力)では実現できません。6(コンソールへの直接入出力)か, BIOS コールを用いて変更する必要があります。)

動作を確認したものは次のものです。

BDS C vl.50a および α-C vl.51 turboCP/M v 2.2 (漢字版) (シャープ)

atokstr(inbuf,outbuf)
char *inbuf,*outbuf;

inbuf で与えられる文字列のうち、1 バイト文字をすべて2 バイト文字に変更して outbuf に書き込みます。従って、outbuf の長さは inbuf の文字列の長さを終端文字0も含めてnとすると、最大2*(n-1)+1となります。元

々 2 バイトの文字は変更しません。 なお、 スペース (20H) はスペース 2 個にします。

また、カ *, ハ °など半角文字では2バイトで1文字になるものも正しく変換されます。ただし、その変換は機械的なものなので、"ア °"などとありえない文字を書くと誤って変換されます。 関数 atok, iskanji を使用しています。

```
int atok(c)
char c;
```

1バイト文字 c を 2 バイト文字に変換し、整数で返します。変換できない場合には 0 を返します。

```
ktoastr(inbuf,outbuf)
char *inbuf,*outbuf;
```

inbufで与えられる文字列で2バイト文字のうち1バイト文字にできるものだけを変更してoutbufに格納します。ただし、スペースが2個あっても1個には変換しませんので、atokstrで変換した文字列をktoastrに入力した場合、スペースについてのみ、異なる可能性があります。

なお、関数 ktoa を用いています。

```
char ktoa(k)
unsigned k;
```

2 バイト文字 k を 1 バイト文字に変換します。変換できない場合は 0 を返します。

関数 iskanji, _ksearch (文字列の検索関数) を用いています。

```
romanstr(str, buf)
char *str, *buf;
```

ローマ字カナ変換を行ないます。この関数は英大文字あるいは小文字で与えられるローマ字の文字列をダイナミックに1バイト文字のカナに変換します。このローマ字変換の規則は次のとおりですが、この規則に当てはまらない場合、ローマ字はそのまま出力バッファにコピーされます。

	□ a	□i	□ u	□ e	□ o
k	カ	丰	7	ケ	コ
s	+	シ	ス	セ	ソ
t	9	チ	"	テ	}
n	+	-	X	ネ	1
h	/\	E	フ	^	ホ
m	7	3	4	×	モ
r	ラ	1)	ル	ν	ロ (]でも変換可能)
у	ヤ	×	ユ	×	Э
w	ワ	ウィ	ウゥ	ウェ	7
g	力 *	+ "	ク・	ケ゛	コ *
Z	# *	シ゛	ス゛	セ"	ソ゛
d	9 "	チ゛	""	テ゛	F "
b	/*	t "	フ゛	~ '	ホ ゛
p	/\ °	E.	フ。	~ °	术 。
f	ファ	フィ	フ	フェ	フォ
v	ウ゛ァ	ウ゛ィ	ウ゛	ウ゛ェ	ウ゛ォ
k y	キャ	キイ	キュ	キエ	キョ
s y	シャ	シイ	シュ	シェ	ショ
t y	チャ	チィ	チュ	チェ	チョ

```
二十 二 二 二 二 二 二 三 3
n y
   ヒャヒィ
           ヒュ ヒェ ヒョ
hy
   ミャ ミィ ミュ ミェ ミョ
m y
    リャ リィ リュ リョ リョ
ry
   キ ヤ キ イ キ ユ キ エ キ ョ
gy
   シ゛ャ シ゛ィ シ゛ュ シ゛ェ シ゛ョ
z y
dy f "+ f "1 f " 1 f " 1
    ヒ * ヤ ヒ * イ ヒ * ユ ヒ * エ ヒ * ヨ
b v
    ヒ。ヤヒ。イヒ。ユヒ。エヒ。ョ
ру
   シ゛ャ シ゛ シ゛ュ シ゛ェ シ゛ョ
i
th テャ ティ テュ テェ テョ
  テ、ヤテ、イテ、ユテ、エテ、ヨ
d h
   ツァ ツィ ツ ツェ ツォ
t s
```

nのみーン

xのみーン

(ウンヨウなどは uxyou とすると正しく変換されます。)

```
htokstr(str,buf)
char *str,*buf;
```

2 バイト文字のひらがなの文字列を 2 バイト文字のカタカナに変換します。 入力文字列の先頭アドレスを str に、出力文字列の出力は buf に設定します。 2 バイトのひらがな以外の文字はそのままコピーします。

```
unsigned htokana(k)
unsigned k;
```

kが2バイト文字のひらがなの場合、それを2バイトのカタカナに直します。もしひらがなでない場合はそのままkの値を返します。

ktohstr(str,buf)
char *str,*buf;

2 バイト文字のカタカナ文字列をひらかなに直します。カタカナ以外の文字が含まれていた場合にはそのまま出力にコピーします。

入力はstr、出力はbufです。

unsigned kanatoh(k)
unsigned k;

2 バイト文字 k がカタカナであった場合にはひらがなに直して戻り値とします。もし、カタカナでなければそのまま返します。

ktype(k)
unsigned k;

2 バイト文字 k の種類を調べ、その種類によって次のような値を返します。

 2バイト文字でない場合……-1

 未定義…
 0

 記号……
 1

 数字……
 2

 英大文字…
 3

 英小文字…
 4

ひらがな5	
カタカナ6	
漢字7	
その他(ギリシャ文字など)8	

```
iskanji(c)
char c;
```

iskanji()は文字 c が 2 バイト文字の 1 バイトめであるかどうかを判定し、そうであれば 1、さもなければ 0 を返します。

これらの関数を含むリスト nihongo. c をリスト 5-3-1 に紹介します. なお, nihongo. c はカナ対応に変換したBDS C, あるいは α -Cでktocを用いてコンパイルしてください。

リスト 5-3-1 nihongo. c

```
/*******************************
1:
2:
             BDS C日本語処理パッケージ v1.0
3:
4:
            Author: Tsu.Mitarai 13/Dec/1986
(c) Armat co. 1986
5:
6:
7:
8:
                /* 利用方法 */
9:
            1.23行の KTEST を D に変更
10:
            2. "nihongo.c" をコンパイル
11:
12:
13:
                    A>ktoc nihongo
                    A>cc nihongo.ck
14:
15:
            3. ターゲットプログラムとリンク
16:
17:
18:
                    A>cc target
                    A>clink target -f nihongo
19:
20:
                    ***********
21:
     #include <bdscio.h>
22:
     #define KTEST 1
23:
24:
25:
     #if KTEST
26:
     main()
27:
28:
            char
                    buf[33];
                    knbuf[33];
29:
            char
                    kbuf [65];
30:
            char
```

```
31:
              char
                       khbuf [65]:
 32:
               char
                       kkbuf [65];
 33:
                      abuf [33];
               char
 34:
               int
 35:
 36:
               while (1)
 37:
 38:
                       printf ("input->"):
 39:
                       n=kgetline (buf, 30);
 40:
                       printf ("オリジナル: Xs (文字数%d) Yn", buf, n);
 41:
 42:
                       romanstr (buf, kbuf);
 43:
                       printf ("ht変換 : %s\n",kbuf);
 44:
45:
                       atokstr (kbuf, knbuf);
 46:
                      printf ("カナー>カナ : %s\n",knbuf);
 47:
 48:
                       ktohstr (knbuf, khbuf);
                      printf ("カナー>かな: %s\n",khbuf,n);
 49:
 50:
 51:
                      htokstr (khbuf,kkbuf);
 52:
                       printf ("かな->カナ: %s\n",kkbuf);
 53:
                       ktoastr (kkbuf,abuf);
 54:
                       printf ("カナー>カナ : %s\n\n",abuf);
 55:
 56:
 57:
 58:
       #endif
 59:
 60:
 61:
      kgetline(buf,cnt)
                                      /* output buffer */
 62:
                    *buf;
              char
 63:
                                      /* 最大文字数 */
              int
                      cnt;
 64:
       {
 65:
              int
                      ccnt:
 66:
              char
                      c;
 67:
              int
                      i:
 68:
              char
                    *pnt;
 69:
 70:
              pnt = buf;
 71:
              ccnt = 0;
              while (1)
 72:
 73:
               {
 74:
                      c=bdos (1);
 75:
                      switch (c)
 76:
 77:
                      case 'Yr':
                                       /* リターンキー */
 78:
                                       *pnt=0;
 79:
                                       bdos (9, "YrYn$");
 80:
                                       return (ccnt);
 81:
 82:
                      case 3:
                                      /* control C */
 83:
                                      exit ():
 84:
 85:
                      case 0x7f:
                                      /* BS, LEFT ARROW, DEL */
 86:
                      case 0x8:
                                       if (ccnt==0)
 87:
 88:
                                               bdos (2,0x1c);
89:
                                              break;
90:
91:
                                       else if (iskanji(*(pnt-2)))
92:
93:
                                              ccnt -= 2;
94:
                                              pnt -= 2;
bdos (9, "\b \b\b\s");
95:
96:
97:
                                       else if (*(pnt-1) < 0x20)
98:
99:
                                              bdos (9, "Yb YbYh$");
100:
                                              ccnt--;
```

```
pnt--;
101:
102:
                                         }
                                         else
103:
104:
                                                 bdos (9, " \b$");
105:
                                                 ccnt--;
106:
                                                 pn t--;
107:
108:
109:
110:
                                         break;
111:
                        case 0x18:
                                         /* contol X */
112:
113:
                                         pnt = buf;
                                         for (i=0; i <ccnt; i++)
114:
115:
                                                 if (*pnt++<0x20) bdos (9,"Yb \byb\b\b");
116:
                                                               bdos (9, " \b\b\s");
                                               else
117:
118:
                                         bdos (9, " \b$");
119:
120:
                                         ccnt = 0;
                                         ont = buf:
121:
122:
                                         break;
123:
124:
                        default:
                                         *pnt++ = c;
125:
                                         ccnt++;
                                         if (c<0x20)
126:
127:
                                         {
                                                 bdos (2,'^');
128:
                                                 bdos (2,c+0x40);
129:
130:
                                         if (ccnt > = cnt-2)
131:
132:
                                                 if (iskanji(c))
133:
134:
                                                  {
                                                          *pnt++ = bdos (1);
135:
136:
                                                          ccnt++:
 137:
                                                 }
                                                 *pnt = 0;
138:
                                                 bdos (9, "Yr Yn$");
 139:
 140:
                                                 return(ccnt);
                                         }
 141:
 142:
                       }
 143:
               }
 144:
      }
 145:
 146:
 147: /* 1 バイト文字列 -> 2 バイト文字列変換 */
      atokstr(str,buf)
 148:
                char *str;
char *buf;
                                       /* input */
 149:
 150:
                                        /* output */
 151:
 152:
                unsigned k;
                while (*str)
 153:
                {
 154:
                         if (*str == '7' && *(str+1) == '"')
 155:
156:
                         {
                                 *buf++ = 0x83;
 157:
                                 *buf++ = 0x94;
 158:
                                 str ++;
 159:
 160:
                        }
                        else if (*str == '"') *(buf-1)+=1;
else if (*str == '"') *(buf-1)+=2;
 161:
 162:
                         else if (iskanji(*str))
 163:
 164:
                         {
                                 *buf++=*str++:
 165:
 166:
                                 *buf++=*str;
 167:
                         else if (k=atok(*str))
 168:
 169:
 170:
                                 *buf++=k/256;
```

```
*buf++=k&0xff:
171:
                      }
172:
                             *buf++=*str:
173:
                      else
174:
                      str++;
175:
              }
176:
              *buf = 0;
     }
177:
178:
179 .
     /* 1 バイト文字 -> 2 バイト文字変換 */
180:
181:
      unsigned atok(c)
182:
              char
183:
184:
              char
              char *s;
if (c<' ')
                             return (0):
185:
              else if (c<0x40)
186:
187:
                      s=" ! "#$%&' () *+, -, /0123456789:; <=>?";
188:
                      c = (c - 0 \times 20) * 2:
189:
190:
              else if (c<0x60)
191:
192:
                      s='@ABCDEFGHIJKLMNOPQRSTUVWXYZ [¥] ^_';
193:
                      c = (c - 0 \times 40) *2;
194:
195 .
              1
              else if (c<0x80)
196:
197:
                      s=" `abcdefghijklmnopqrstuvwxyz {|} ~ ";
198:
199:
                      c = (c - 0 \times 60) *2;
200:
              }
              else if (c<0xa0)
                                   return (0);
201:
              else if (c<0xc0)
202:
203:
                      s=" 。「」、・ヲァィゥェオャュョッーアイウエオカキクケコサシスセソ";
204:
205:
                     c = (c - 0xa0) *2;
206:
              }
              else if (c<0xe0)
207:
208:
                      s="タチツテトナニヌネノハヒフヘホマミムメモヤユヨラリルレロワン' ° ":
209:
                      c = (c - 0 \times c0) *2:
210:
              }
211:
212:
              else
                     return (0);
              return (*(s+c)*256 + *(s+c+1));
213:
214:
215:
216:
      /* 2 バイト文字列 -> 1 バイト文字列変換 */
217:
218:
      ktoastr(str, buf)
                             /* input */
219:
              char
                     *str:
220:
              char
                     *buf;
                            /* output */
221:
      -{
222:
              unsigned c:
              while (*str)
223:
224:
                      if (c = ktoa (*str*256+*(str+1), 1))
225:
226 .
                              if (c<256)
                                             *buf++ = c;
227:
228:
                              else
229:
                              {
                                      *buf++ = c & 0xff;
230:
                                      *buf++ = c / 256;
231:
232:
                              }
233:
                              str+=2;
234:
                      else if (iskanji (*str))
235:
236:
                              *buf++ = *str++;
*buf++ = *str++;
237:
238:
239:
                      else
240:
```

```
2/11:
                    {
                           *buf++ = *str++;
242:
243:
244:
245:
             *buf = 0:
246:
247:
248:
249:
     /* 2 バイト文字 -> 1 バイト文字変換
         湯音、半濁音の場合には上位バイトに'*', '*'
250:
         のコードが設定されます.
251:
252:
253:
     unsigned ktoa(kn)
254:
           unsigned kn;
255:
256:
             char
257:
             unsigned k;
258:
             if (!iskanii(kn/256)) return (0);
259:
260:
            s=' ! " \sharp $ % & ' () * +, -. / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?"; if (ERROR != (k=_ksearch (s,kn)))
261:
262:
263:
                    return (k+0x21);
264:
265:
             s= "@ABCDEFGHIJKLMNOPQRSTUVWXYZ [¥] ^_";
266:
             if (ERROR != (k=_ksearch (s,kn)))
                    return (k+0x40):
267:
268:
             s=" `abcdefghijklmnopqrstuvwxyz (|) ~ ";
269:
270:
             if (ERROR != (k=_ksearch (s,kn)))
271:
                    return (k+0x60);
272:
             273:
274:
                    return (k+0xal);
275:
276:
             s= *タチツテトナニヌネノハヒフへホマミムメモヤユヨラリルレロワン * * *;
277:
             if (ERROR != (k=_ksearch (s,kn)))
278:
279:
                    return (k+0xc0);
280:
             s= "ガギグゲゴザジズゼゾダヂヅデド";
281:
282:
             if (ERROR != (k=_ksearch (s,kn)))
                                                 return ('"' *256+k+0xb6);
283:
             s= "バビブベボ":
284:
             if (ERROR != (k=_ksearch (s,kn)))
                                                return ('"' *256+k+0xca);
285:
286:
287:
             s="パピアペポ":
288:
             if (ERROR != (k=_ksearch (s,kn)))
                                                return ('°'*256+k+0xca);
289:
             if (k=='ヴ')
290:
                                                 return ('"7');
291:
            return (0);
292:
293:
294:
295:
296:
297:
     /* 2 バイト文字列のサーチ */
298:
     _ksearch(str,k)
299:
             char
                  *str;
300:
             unsigned k;
301:
      {
302:
             char
                   c0.c1;
303:
                   i;
             int
304:
305:
             i = 0;
             c0 = k/256;
306:
307:
             cl = k&0xff;
308:
             while (1)
309:
             {
310:
                     if ((*str == c0) && (*(str+1) == c1)) return (i);
```

```
if (!*(++str)) return (-1);
311:
                      if (!*(++str)) return (-1);
312:
313:
                      i++;
             }
314:
315:
316:
317:
      /* ローマ字文字列->ht文字列変換 */
318:
319:
     romanstr(str, buf)
                             /* input */
320:
              char
                      *str:
                             /* output */
                      *buf;
321:
              char
322:
      {
323:
              char
                    kbuf [5];
              *buf = 0:
324:
325:
              while (*str)
326:
              {
327:
                      str=romatok (str,kbuf);
328:
                      strcat (buf, kbuf);
              }
329:
330:
     }
331:
332:
      /* ローマ字 カナ(1パイトモラ゚) 変換 */
333:
334:
     romatok (pnt, buf)
                      *pnt:
335:
              char
336:
                      *buf;
              char
      {
337:
338:
              char
                      c:
                      c1,c2,c3;
339:
              char
340:
              int
                      ku;
341:
              char
                      *str;
342:
343:
              str = pnt;
              cl = toupper (*str++);
3/4:
345:
              346:
347:
348:
               else
349:
               {
350:
                      c2 = toupper (*str++);
                      if (c=_ckboin (c2))
351:
352:
                              switch (c1)
353:
354:
                              case 'K':
                                             *buf++=c+5;
355:
356:
                                              break;
                            case 'S':
357:
                                              *buf++=c+10;
358:
                                              break;
                            case 'T':
                                              *buf++=c+15;
359:
360:
                                              break:
                            case 'N':
                                             *buf++=c+20;
361:
362:
                                              break;
                              case 'H':
                                              *buf++=c+25;
363:
364:
                                              break:
                              case 'M':
                                              *buf++=c+30;
365:
                                              break;
366:
                              case 'L':
367:
                              case 'R':
                                             *buf++=c+38;
368:
                                              break:
369:
                              case 'G':
370:
                                              *buf++=c+5;
371:
                                              *buf++='"';
                                              break:
372:
                              case 'Z':
373:
                                              *buf++=c+10;
374:
                                              *buf++= " " ;
375:
                                              break:
376:
                              case 'D':
                                              *buf++=c+15;
377:
                                              *buf++='"';
378:
                                              break:
                              case 'B':
379:
                                              *buf++=c+25;
                                              *buf++='"';
380:
```

```
381:
                                                   break:
                                  case 'P':
382:
                                                   *buf++=c+25;
383:
                                                   *buf++=' ° ';
384:
                                                   break;
                                  case 'Y':
                                                   if (c=='7')
385:
                                                                     *buf++=' ";
                                                   else if (c=='7') *buf++='1';
else if (c=='7') *buf++='3';
386:
387:
388:
                                                                     goto kerror;
                                                   else
389:
                                                   break:
390:
                                 case 'W':
                                                   if (c=='P')
                                                                     *buf++='7';
                                                   else if (c=='7') *buf++='7';
391:
392:
                                                   else
393:
394:
                                                            *buf++='9';
395:
                                                            *buf++=c-10;
396:
397:
                                                   break;
398:
                                 case 'F':
                                                   if (c=='5')
                                                                   *buf++='7';
399:
                                                   else
400:
401:
                                                           *buf++='7';
402:
                                                           *buf++=c-10;
403:
404:
                                                   break;
                                                   *buf++='3';
                                 case 'J':
405:
                                                   *buf++='"';
406:
407:
                                                   c=_ckyoon (c);
                                                   if (c!='4')
408:
                                                                    *buf++=c;
409:
                                                   break;
410:
                                 case 'V':
                                                   *buf++='?':
                                                   *buf++='"
411:
412:
                                                   *buf++=c-10;
413:
                                                   break;
414:
                                 default:
                                                   goto kerror;
415:
416:
                         else if (cl != 'N' && c1==c2)
417:
418:
419:
                                 *buf++='9';
420:
                                 str--;
421:
422:
                         else if (c1 == 'N' && (c1==c2 || c2==0 || c2!='Y'))
423:
                         {
424:
                                 *buf++='>':
425:
                                 str--;
426:
                         }
427:
                         else
428:
429:
                                 c3 = toupper (*str++);
                                 if (!(c=_ckboin(c3))) goto kerror;
430:
431:
432:
                                 c = _ckyoon (c);
433:
                                 ku=c2*256+c1;
434:
                                 switch (ku)
435:
436:
                                 case 'KY':
                                                  *buf++=' *':
437:
                                                   *buf++=c;
438:
                                                   break;
439:
                                 case 'SY':
                                                   *buf++='5';
440:
                                                   *buf++=c;
441:
                                                   break;
                                                  *buf++=' +':
442:
                                case 'TY':
443:
                                                   *buf++=c;
444:
                                                   break;
445:
                                 case 'NY':
                                                   *buf++='=';
446:
                                                   *buf++=c;
447:
                                                   break:
448:
                                                  *buf++='t';
                                 case 'HY':
449:
                                                   *buf++=c;
450:
                                                  break:
```

```
451:
                                 case 'MY':
                                                   *buf++='3';
452:
                                                   *buf++=c;
453:
                                                   break;
                                                   *buf++='";
                                 case 'RY':
454:
455:
                                                   *buf++=c;
                                                   break:
456:
                                                   *buf++=' +';
457:
                                 case 'GY':
                                                   *buf++='"';
458:
459:
                                                   *buf++=c:
460:
                                                   break;
                                 case 'ZY':
                                                   *buf++='5';
461:
                                                   *buf++='"';
462:
                                                   *buf++=c;
463:
464:
                                                   break;
                                 case 'DY':
                                                   *buf++=' 7';
465:
                                                   *buf++='"';
466:
467:
                                                   *buf++=c:
468:
                                                   break;
                                 case 'BY':
                                                   *buf++='t';
469:
                                                   *buf++='"';
470:
                                                   *buf++=c;
471:
                                                   break;
472:
                                 case 'PY':
                                                   *buf++='t':
473:
                                                   *buf++=' *';
474:
475:
                                                   *buf++=c:
476:
                                                   break:
                                                   *buf++='5';
                                 case 'SH':
477:
                                                   if (c!='4')
478:
                                                                    *buf++=c;
479:
                                                   break:
                                                   *buf++='f';
if (c!='4')
480:
                                 case 'CH':
481:
                                                                    *buf++=c;
                                                   break;
482:
                                                   *buf++='";
                                 case 'TS':
483:
                                                   if (c!='a')
484:
                                                                   *buf++=c:
485:
                                                   break;
                                                   *buf++='.7';
486:
                                 case 'TH':
487:
                                                   *buf++=c;
488:
                                                   break:
                                 case 'DH';
                                                   *buf++='7';
489:
                                                   *buf++='"';
490:
491:
                                                   *buf++=c;
192.
                                                   break:
                                                   if (cl=='N')
                                 default:
493:
494:
                                                            *buf++='>';
495:
                                                            str -= 2;
496:
                                                   1
497:
498:
                                                   else
                                                           goto kerror;
499:
                                 }
500:
                        }
501:
502:
                *buf=0;
                return (str);
503:
504:
      kerror: *buf++=*pnt++;
505:
506:
                *buf = 0;
                return (pnt);
507:
508:
       }
509:
510:
       /* 母音のチェック */
511:
       _ckboin(c)
512:
                char c;
513:
514:
515:
                switch (c)
516:
                case 'A':
case 'I':
case 'U':
case 'E':
                                 return ('?');
517:
                                 return ('1');
518:
519:
                                 return ('');
                                 return ('I');
520:
```

```
case '0':
                          return ('†');
return (0);
521:
522:
              default:;
523:
524:
     }
525:
526:
     /* よう音のチェック */
527:
     _ckyoon(c)
528:
              char c;
                            /* ht */
529:
530: {
531:
              switch (c)
532:
              {
533:
              case 'P':
                             return ('r');
                           return ('4');
              case '1':
534:
             case '7':
case 'I':
case '7':
                           return ('x');
return ('x');
535:
536:
537:
                             return ('a');
              default:;
538:
539:
540:
              return (c);
541: }
542:
543:
     /* ひらがな文字列 ->カタカナ文字列変換 */
544:
545: htokstr(str,buf)
546:
              char *str;
547:
              char
                    *buf;
548:
      {
549:
              unsigned k;
550:
              while (1)
551:
              {
                      if (!*str)
                                             break;
552:
553:
                      if (!iskanji (*str)) *buf++=*str++;
554:
                      else
555:
                      {
556:
                              k= *str++;
                              if (!*str)
                                                    break:
557:
                              k=k*256+*str++;
558:
                              k=htokana (k);
559:
                              *buf++=k/256;
560:
561:
                              *buf++=k & Oxff;
562:
563:
564:
              *buf = 0;
565: }
566:
567:
     /* ひらがな->カタカナ変換 */
568:
569: unsigned htokana(k)
570:
              unsigned k;
571:
              if (ktype (k) == 5)
572:
573:
574:
                      if (k>0x82dd) return (k+0xa2);
575:
                                     return (k+0xal);
                     else
576:
              }
577:
              return (k);
578:
579:
     /* カタカナ文字列 ->ひらがな文字列変換 */ktohstr(str,buf)
580:
581:
              char *str;
char *buf;
582:
583:
      {
584:
585:
              unsigned k;
586:
              while (1)
587:
              {
                                            break;
588:
                      if (!*str)
589:
                      if (!iskanji (*str)) *buf++=*str++;
590:
                      else
```

```
{
591:
592:
                             k=*str++;
593:
                             if (! *str)
                                                   break;
                             k=k*256+*str++;
594:
595:
                             k=kanatoh (k);
                             *buf++=k/256:
596:
                             *buf++=k & Oxff;
597:
                     }
598:
599:
600:
             *buf=0;
601: }
602:
603:
     /* カタカナ->ひらがな変換 */
604:
     unsigned kanatoh(k)
605:
             unsigned k;
606:
607:
      {
             if (ktype (k) == 6)
608:
609:
610:
                     if (k>0x837f)
                                    return (k-0xa2);
                                    return (k-0xal);
611:
                     else
612:
             return (k);
613:
614: }
615:
616:
      /* 2パイト文字の種類をチェックする関数
617:
618:
      出力: -1 (2バイト文字ではない)
               ① (未定義)
619:
               1 (記号)
620:
               2 (数字)
621:
               3 (英大文字)
622:
               4 (英小文字)
623:
624:
               5 (ひらがな)
               6 (カタカナ)
625:
626:
               7 (漢字)
              8 (ギリシャ文字など)
                                        */
627:
628:
629:
     ktype(k)
             unsigned k;
630:
631:
      {
              char kl;
632:
              kl = k&Oxff;
633:
634:
             if (!iskanji (k/256))
                                   return (-1);
635:
                                    return (0);
636:
              if (k1<0x40)
             if (kl = 0 \times 7f)
637:
                                    return (0);
                                    return (0);
638:
              if (kl>0xfc)
639:
                                    return (0);
             if (k<0x8140)
640:
              if (k<0x824f)
if (k<0x8258)
641:
                                    return (1);
                                    return (2);
642:
643:
              if (k<0x8260)
                                    return (0);
             if (k<0x827a)
                                    return (3);
644:
              if (k<0x8281)
                                    return (0);
645:
646:
              if (k<0x829a)
                                     return (4);
              if (k<0x829f)
                                    return (0);
647:
              if (k<0x82f2)
                                    return (5);
648:
              if (k<0x8340)
                                    return (0):
649:
650:
              if (k<0x8397)
                                    return (6);
651:
              if (k<0x849f)
                                    return (8);
652:
              return (7);
653: }
654:
655:
656:
      /* 2パイト文字列かどうかのチェック */
      iskanii(c)
657:
             char c:
658:
659:
              if (c<0x80) return (FALSE);
660:
```

```
661: if (c<0xa0) return (TRUE);
662: if (c<0xe0) return (FALSE);
663: if (c<0xfd) return (TRUE);
664: return (FALSE);
665: }
```

5-4 日本語処理パッケージの使い方

nihongo.c はそのままではテスト用の関数を含んでいますので23行めの#define文のktestを 0 に変更してコンパイルし、nihongo.crl というファイルにしておきます。これを用いてプログラムを作成する場合には、

clink program -f nihongo

という形で使うと自動的に nihongo. crl から必要関数だけをピックアップして 使うことが可能です.

また、BDS Cであらかじめ用意されている文字列処理用の関数のうち、日本語をうかつに使うと問題になるものがあります。toupper、tolowerがその代表的なもので、この関数を用いて文字列などを変換すると漢字コードの2バイトめが英大文字あるいは小文字と同じコードのものはすべて異なった文字に変換されてしまいます。そのためあらかじめ2バイトコードでないことを確認して使う必要があります。

なお、ktocの処理は決して速いとは言えません。そこで、日本語の文字列がある部分だけを別のファイルにしておき、そこだけktocで変換しても良いでしょう。日本語の無い部分までktocに通す必要はありません。

操作例 5-4-1 日本語処理パッケージの操作例

A><u>ktoc nihongo</u>← 日本語処理パッケージのコンパイルにはktocが必要。 **<NIHONGO.CK>を作成しました**

A>cc nihongo.ck → コンパイルにも漢字対応に変更したものを使う. BD Software C Compiler v1.50a (part I) 21K elbowroom BD Software C Compiler v1.50 (part II) 23K to spare

A>clink nihongo
BD Software C Linker v1.50
Linkage complete
36K left over

A>nihongo -テストプログラムの実行

input->akasatanahamayarawan

オリジナル : akasatanahamayarawan (文字数20)

input->fikyunyahyuryajathidhitsu

オリジナル: fikyunyahyuryajathidhitsu (文字数25) カナ変換: フィキュニャヒュリャシ ** ャティテ ** ィリ ◆ ** 助音も正しく変換できる。

ht->カナ : フィキュニャヒュリャジャティディツカナ->かな : ふぃきゅにゃひゅりゃじゃてぃでぃつかな->カナ : フィキュニャヒュリャジャティディツ

カナー>カナ : フィキュニャヒュリャシ"ャティテ"ィツ

input->ro-ma字 カタカナ カタカナ

オリジナル: ro-ma字 hタhナ カタカナ (文字数21)

カナ変換: ローマ字 カタカナ カタカナー ローマ字以外の部分はそのまま.

ht->カナ : u-v字 カタカナ カタカナ カナ->かな : S-ま字 かたかな かたかな かな->カナ : u-v字 カタカナ カタカナ

カナー>カナ : ローマ字 カタカナ カタカナ

input->漢字 E 問題 nashi.

オリジナル: 漢字 t 問題 nashi. (文字数18)

カナ変換 : 漢字 モ 問題 ナラ.

ht->カナ : 漢字 モ 問題 ナシ. カナ->かな : 漢字 も 問題 なし. かな->カナ : 漢字 モ 問題 ナシ. カナ->ht : 漢字 モ 問題 ナシ. input->^C A>

以上、日本語処理についてかなり詳細に述べました。BDS \$, α-Cで日本語処理はできないと思っていた方々には参考にしていただけると思います。また、最低限これだけでも機能が揃うと新しいプログラムの可能性も生れてきます。是非、新分野に挑戦していただきたいと思います。

第 6 章 オーバーレイとチェイン

BDS C. α -Cによって、本格的なプログラムを作成してくるようになる と、やがて問題になってくることがあります。

その1つはCP/M(および8ビットCPUの)最大の欠点である。メモリ サイズの問題で、プログラムが大きくなり過ぎたため、プログラムがす ベてメモリに入り切らなくなった場合です。これには本質的な手だては ありません. あるとすれば、CPUを十分なメモリを確保できる16ビット 以上のものに交換することでしょう.

勿論これは現実的ではありませんから、実際にはプログラムをいくつ かに分割し、必要とする際にそのプログラムをディスクからロードする オーバーレイの技法が良く用いられます。本章ではBDS C特有のテクニ ックであるオーバーレイとチェインについて解説します.

6-1 プログラムのチェイン

プログラムが大きくなり、どうしてもメモリ上にすべてが入りきらなくなってきた場合、ディスクからプログラムを呼び出しながら実行するオーバーレイ技法を必要とするようになります。これは非常に大きなプログラムでなくとも、作業用のワークメモリエリアを大きく確保したい場合にも用いられることがあります。

実行中のプログラムがその一部を切り放して別のプログラムモジュールをディスクからロードして実行を継続するオーバーレイに対し、実行中のプログラムすべてを捨て去り、全く新しいプログラムをディスクからロードして実行させることをチェインと言います(BASICではMERGE、およびRUN "filename" に当るでしょうか.).

BDS Cでは、オーバーレイは比較的簡単にできるように設計されており、いくつかの手順を踏むだけで実現できます。ただし、プログラムをディスクから呼び出しながら実行していくため、プログラムを機能単位で分割し、あらかじめ整理することがどうしても必要です。単純にプログラムが大きくなり過ぎたから、というだけではオーバーレイ構造をとることはできません。しかし、別個の実行ファイル(comファイル)を作るチェインで構わないのであれば、もっと楽に作成することができます。BDS C, α -Cにはexec, execv, excel x どチェイン用の関数が用意されていますので、作成は簡単です。

この方法は、プログラムのランタイムパッケージの部分がダブってロードされるため、ディスク容量、ファイルロード時間などの点で不利ですが、全く別のプログラムとしてデバッグできるため、非常に簡便で、いつでも実現できる方法だと言えます。

BDS Cの場合指定しない限り外部変数はすべてクリアされてしまいますので、メモリ上のデータを渡すことはできませんが、execv, execlの場合はコマンド行からの引数を渡すこともできますから最低限の情報を送ることは可能です。

リスト6-1-1 execv関数によるチェイン

```
A>type execroot.c
#include <bdscio.h>
main (argc, argv)
                                    int
                                                                          argc;
                                     char
                                                                          *argv[];
                                                                                                                                                                                                                    呼び出し側の
                                                                                                                                                                                                                     プログラム.
                                    printf ("Root segment excuting. Yn");
                                    argv[argc]=0;
                                    execv ("execov1.com", &argv[1]);
}
                                                                                                                                                                                   呼び出される側のプログラム.
                                                                                                                                                                                   これはリスト 2-2-1 の args.c とほとんど同じもの.
A>type execovl.c
                                                                                                                                                                                   単体で実行できる。
#include <bdscio.h>
main (argc, argv)
                                     int
                                                                          argc;
                                                                          *argv[];
                                     int
                                                                       i;
                                     printf ("Execovl excuting....\fm");
                                     for (i=1; i<argc; i++)
                                                                          printf ("arg%d=\footnotesis"\footnotesis \footnotesis \footnotesi
}
A>cc execroot
BD Software C Compiler v1.50a (part I)
         35K elbowroom
BD Software C Compiler v1.50 (part II)
         32K to spare
A>clink execroot
BD Software C Linker
                                                                                                      v1.50
Linkage complete
```

42K left over

A>cc execovl

BD Software C Compiler v1.50a (part I) 35K elbowroom BD Software C Compiler v1.50 (part II) 32K to spare

A>clink execovl
BD Software C Linker v1.50
Linkage complete
43K left over

A>execroot a bc def ghij

Root segment excuting.

Execov1 excuting....

arg1="A"
arg2="BC"
arg3="DEF"
arg4="GHIJ"

executing.

A>

なお、呼び出される側のプログラムのリンク時に-zオプション(外部変数 クリア禁止)を指定し、かつ呼び出し側のプログラムと外部変数の順番、先頭アドレスを揃えておくとすべての変数を受け渡すことが可能です。ただし、BDS C付属のL 2 リンカー(α -Cにはありません)では-zに相当するオプションがないので、生成されたCOMファイルをDDT などでパッチを当て、禁止するしか方法はありません(141番地を $C3H \rightarrow C9H$ に変更する)。

また、execv, execl などの関数では、呼び出すのはCP/Mのユーティリティ である PIPやASMあるいはCコンパイラでも構わないという利点もあります (但し、DIR、ERA などのビルトインコマンドは使えません。).

BDS Cのコンパイラ本体である cc. com もこの execv, execl などからコマンドラインの文字列を設定して実行させることが可能であり、そうすると

ktocのようなプリプロセッサも利用が簡単になります.

そこで、ktocのリストのmain 関数だけを変更して自動的にコンパイラを起動する ck. com というプログラムに作り換えてみましょう。ck に対するオプションはそのまま cc. com に対するオプションとなりますので、nk (カタカナのみ) オプションはなくなっています。

リスト 6-1-2 ck. cと操作例

```
main (argc, argv)
int
       argc:
char
       *argv[];
               infile[20]:
       char
       char
               outfile[20]:
       if (argc < 2)
               printf ("書式: ck filename [options] \n"); ←
               exit ();
       strcpy (infile,argv[1]):
       strcpy (outfile, argv[1]);
       strcat (infile, ".C");
       streat (outfile, ".CK");
       kanjif = TRUE;
       if (ERROR == fopen (infile, fpin))
               printf ("<%s>がオープンできません", infile);
               exit ():
       if (ERROR == fcreat (outfile, fpout))
               printf ("<%s>を作成することができません",outfile);
               exit ():
                             /* メインルーチン */
       ktoc ():
       fputc (CPMEOF);
       if (ERROR == fclose (fpout))
               printf (*<%s>を正しくクローズできません*,outfile);
       else
               printf ("<%s>を作成しました¥n",outfile);
               argv[1] = outfile;
                                                     変更
               argv[argc]=0;
               execv ("cc.com", &argv[1]);
               (以降変更なし)
```

```
A>cc\ ck\ -e2200 \leftarrow execv 関数が加わるためサイズが大きくなる。-e2200 でコンパイル。
BD Software C Compiler v1.50a (part I)
  32K elbowroom
BD Software C Compiler v1.50 (part II)
  29K to spare
A>clink ck
BD Software C Linker v1.50
Linkage complete
  38K left over
A>ck ck -e2200 -p ← 完成した ck. com で自分自身をコンパイルしてみる.
<CK.CK>を作成しました
BD Software C Compiler v1.50a (part I) - ck.comにより自動的に
                                              コンパイルが起動される.
        :(前略)
  24: main(argc, argv)
  25: int
  26: char
               *argv[];
  27: {
  28: char
             infile[20];
  29: char outfile[20];
  30:
                                            このように変換されているため.
  31: if (argc < 2)
                                            入力ファイルは正しく ck.ck である、
  32:
  33:
               printf ("\217\221\216\256: ck filename [options]\n");
  34:
               exit ():
  35:
       }
        (中略)
                      printf ("\225\266\216\232\227\361\202\252\222\2267\211
¥337¥202¥254¥202¥351¥202¥251¥°¥203}¥201;¥203N¥202¥252¥227¥216¥202¥277¥202¥304
¥202¥242¥202¥334¥202¥267°);
 196:
                      exit ();
 197:
               }
 198:
              else fputc (*pnt++);
 199: }
 200: }
  31K elbowroom
BD Software C Compiler v1.50 (part II)
  29K to spare
A>clink ck
BD Software C Linker v1.50
Linkage complete
  38K left over
```

なお、このckを利用する場合、cc. com は必ずデフォルトドライブ (A) の

時にはドライブA)に入れておく必要があります。

また, main 関数の最後で

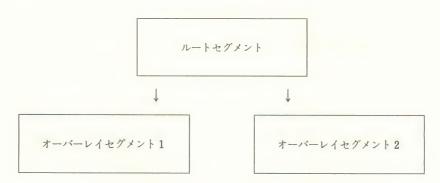
argv[argc]=0;

という式がありますが、ポインタ配列rgvは0からrgc-1までの要素しか存在していないので、本来このような使い方をしてはいけません。しかし、BDS C、 α -C ではランタイムパッケージの部分にあらかじめ30個までrgv 用の空間が確保されており、問題が発生することはありません。

6-2 swapin関数を用いたオーバーレイ

さて、プログラムが長くなり、実行時に必要なプログラムモジュールをロードする本格的なオーバーレイをとらねばならなくなった場合、あらかじめ プログラムを次のような観点から、機能ごとに振り分ける必要がでてきます。

- 1. 常に存在しなければならない関数 (機能)
- 2. 必要に応じて実行できれば良い関数 (機能)
- 1. はオーバーレイする際の常駐プログラム (ルートセグメント)となり、2がオーバーレイセグメントになります。このオーバーレイセグメントを複数用意し、どのプログラムをロードするかによって実行できる機能を切り換えます。



図でお判りのように、オーバーレイセグメント 1 が存在しているときにはオーバーレイセグメント 2 の関数は実行することはできません。両者共通に必要な関数はルートセグメントに入れておくか、両方のファイルに同じように入れておかなければなりません。

また、これはコンパイルしてから言えることですが、ルートセグメントのサイズそのものが大きくなり過ぎると共用メモリ領域が狭くなり、オーバーレイセグメントが入り切らなくなります。特に、1~2 Kバイトしかない空間をいくつものオーバーレイセグメントで共用するという方法は余りにも煩雑であり、うまいやり方とは言えないでしょう。このような場合には関数のレイアウトを考慮して4 Kバイト程度のメモリをまず確保し、その上で、オーバーレイセグメントは2つなり3つなりのファイルに厳選するほうが良いと思います。この時、オーバーレイセグメント1と2が大体同じようなサイズになると、効率も良くなります。

オーバーレイセグメントの読み込みはルートセグメントから行ないます. この時,このファイル名を ovlprg1.comとし,読み込むアドレスを 0 x4000とすると,

swapin("ovlprgl.com",0x4000);

としてメモリ上にロードする方法が良く用いられます。0x4000 はそのままオーバーレイセグメントを呼び出す時のアドレスでもあります。 swapin 関数

は簡単にディスク上のファイルの内容をメモリに呼び出すことができるので 便利ですが、ファイルの長さをチェックしないので注意しなければならない 場合もあります。

オーバーレイ関数の実行は関数へのポインタを用いて間接的に行ないます。

```
int (*ovlprg) ();
:
:
ovlprg=0x4000;
(*ovlprg) (引数);
```

という形になります。上記の方法はオーバーレイ時に限らず、関数名ではなくアドレスで関数呼出しを実行する場合の一般的な方法です。

それでは実際の例をご紹介します.

リスト6-2-1 オーバーレイのサンプルと操作例

A>type overlay.c

```
int (*ovlprg) ();
        if (ERROR == swapin ("ovll.com", 0x4000))
                printf ("ovll overlay error !\n");
        else
                                           -swapinにより
                                           オーバーレイセグメント
        {
                ovlprg = 0x4000;
                (*ovlprg) (); ←
        }
}
ov12()
                (*ovlprg) ();
        int
        if (ERROR == swapin ("ov12.com", 0x4000))
                printf ("ov12 overlay error !\n");
        else
                ovlprg = 0x4000;
                (*ovlprg) ();
}
A>type ovll.c
#include <bdscio.h>
main()
        printf ("it is in ovll\n");
}
A>type ov12.c
#include <bdscio.h>
main()
{
```

```
printf ("it is in ov12\fm");
}
                                    -レイ時には外部変数とコードが
                                ダブらないように外部変数アドレスを
A>cc overlay -e6000 -
BD Software C Compiler v1.50a (part I)
  34K elbowroom
BD Software C Compiler v1.50 (part II)
  31K to spare
A>clink overlay -w
BD Software C Linker
                       v1.50
Linkage complete
  42K left over
A>cc ov11 -e6000
BD Software C Compiler v1.50a (part I)
  35K elbowroom
BD Software C Compiler v1.50 (part II)
                                          オーバーレイセグメント指定.
  32K to spare
                                      -14000 実行開始アドレスを
                                           4000Hに指定。
                                      -y overlay
A>clink ovl1 -v -14000 -y overlay
                                           overlay.sym という
                                           シンボルファイルを読込
み、既に存在する関数は
BD Software C Linker
                                           リンクしない.
Linkage complete
  46K left over
A>cc ov12 -e6000
BD Software C Compiler v1.50a (part I)
  35K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
A>clink ov12 -v -14000 -y overlay
BD Software C Linker
Linkage complete
  46K left over
```

A>overlay

```
Which function (1 or 2)..\frac{1}{2} it is in ovl1 — \frac{1}{2} \frac{1}{2} \frac{1}{2} \frac{1}{2} \frac{1}{2} Which function (1 or 2)..\frac{1}{2} it is in ovl2 Which function (1 or 2)..\frac{1}{2} it is in ovl1 Which function (1 or 2)..\frac{1}{2} it is in ovl2 Which function (1 or 2)..\frac{1}{2} \frac{1}{2} Which function (1 or 2)..\frac{1}{2}
```

リスト 6-2-1 は小さい 2 つのオーバーレイセグメントを 4000H 番地から読み込み,それと同時に実行させている例です.1 を選ぶと ovl1. com を読み込みます.実行はロードアドレスと同じく 4000 H 番地からです.

ここで重要なことはこのオーバーレイセグメントの中にはメインプログラム中に存在するprintfなどの関数が含まれておらず、オーバーレイセグメントからのprintf関数の呼出しはメインプログラム内のサブルーチンをコールすることで行なっているということです。これらの処理は自動的に行なわれますので、頭を悩ませることはありません。

6-3 オーバーレイ関数の呼び出し方法

さて、前項の例はオーバーレイモジュール1つにつき1つの関数だけでしたが、実際にこのような場合に問題になるのは次の2点です。

- 1. モジュールが既にロードされているかどうかの確認をどうするか.
- 2. モジュールの中に複数の関数がある場合、どうやって呼び出しを 行なうか。

実行前に必ずファイルをロードしてやれば問題はありませんが、オーバーレ

イ時はかなりディスクアクセスに時間がかかりますから、もし既に必要なセグメントがメモリ上に存在しているなら、ロードせずに済ませたいものです。また、ルートセグメント(常駐のメインプログラム)からはオーバーレイセグメントの先頭アドレス以外は知ることができませんので、関数が2つ以上ある場合、2番め以降の関数をダイレクトに呼び出すことができません。

そこで、手続は少し面倒になりますが、オーバーレイセグメントの1番最初に、あらかじめチェック用の関数を記述しておき、現在存在しているモジュールの番号を返すようにしておきます。すると、必要な関数を呼び出す際に、まずメモリ上にあるかどうかをテストして、なければディスクからロード、あればそのまま関数を呼び出せば良いわけです。また、この関数を引数により複数の関数を呼び出す複合関数にしておけば、呼び出しアドレスが判らなくとも構わないことになります。

具体的には次のように行ないます.

リスト6-3-1 オーバーレイのチェック

```
A>type overlay2.c
#include <bdscio.h>
#define FUNCO
                0
#define FUNC1
#define funca() func(1)
                         関数funca ~ funcdをマクロ定義している。このように
#define funcb() func()(2)
                         すると引数の異なる同一関数の呼び出しが異なる関数
#define funcc() func1(1)
                         の呼び出しのように扱える.
#define funcd() func1(2)
main()
                buf[4]:
        if (ERROR == swapin ("func0.com", 0x4000)) → 起動時に何か
                                                    ルを読み込ん
                printf ("No overlay segment.");
                                                     でおくことが
                                                    必要。
                exit ():
        while (1)
```

```
{
                 printf ("Which function (1-4)..");
                 getline (buf, 3);
                                          funca (); メインルーチンではオーバ
                         (*buf=='1')
                                          funcb (); ーレイを意識
funcs (): せず, 通常の
                 else if (*buf=='2')
                                          funcc (); 関数呼び出し
                 else if (*buf=='3')
                                          funcd (); のように扱え
                 else if (*buf=='4')
        }
}
func((n)
        int
                 n:
               (*ovlprg) ();
        ovlprg = 0x4000;
        if ((*ovlprg)(①)!= FUNCO)→メモリ上にあるセグメントがfuncO.com
                                        でなければファイルをロードする.
                 if (ERROR == swapin ("func0.com", 0x4000))
                         printf ("func0 overlay error !\fm");
                         return:
        (*ovlprg) (n); ←
                              -引数を持ってオーバーレイセグメント
                              上の関数を実行する
}
func1(n)
        int
                n;
{
        int
                (*ovlprg) ();
        ovlprg = 0x4000;
        if ( (*ovlprg) (0) != FUNC1)
                 if (ERROR == swapin ("func1.com", 0x4000))
                         printf ("funcl overlay error !\"n");
                         return:
        (*ovlprg)(n);
}
```

```
#include <bdscio.h>
#define FUNCO
main(n)
        char
               n;
        switch (n)
                                        オーバーレイセグメント種類の
チェック.当然、ファイル毎に
戻り値は変える必要がある。
        case 0: return (FUNCO);
        case 1: funca ();
                return;
        case 2: funcb ();
                 return;
        default:;
}
funca()← 実際に実行したい関数funca
        printf ("This is function A\f\n");
funcb()
        printf ("function B excuting. \f");
}
A>type funcl.c
#include <bdscio.h>
#define FUNC1 1
main(n)
        char
                n:
{
        switch (n)
        case 0: return (FUNC1);
        case 1: funcc ();
                 return;
         case 2: funcd ();
                 return;
```

```
default:;
}
funcc()
        printf ("This is function C\f");
}
funcd()
       printf ("function B excuting. \f");
}
〈ルートセグメントのコンパイル〉
A>cc overlay2 -e6000
BD Software C Compiler v1.50a (part I)
  34K elbowroom
BD Software C Compiler v1.50 (part II)
  31K to spare
A>clink overlay2 -w
BD Software C Linker
                      v1.50
Linkage complete
  42K left over
〈オーバーレイセグメント 0 のコンパイル〉
A>cc func0 -e6000
BD Software C Compiler v1.50a (part I)
  35K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
A>clink func0 -v -14000 -y overlay2
BD Software C Linker v1.50
Linkage complete
  46K left over
〈オーバーレイセグメント1のコンパイル〉
A>cc func1 -e6000
BD Software C Compiler v1.50a (part I)
  35K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
```

A>clink func1 -v -14000 -y overlay2
BD Software C Linker v1.50
Linkage complete
46K left over

A>overlay2

Which function (1-4)...1This is function A Which function (1-4)...2function B excuting. ファイル読出しが行なわれる. Which function (1-4)..3: This is function C ディスクアクセスなし. Which function (1-4)...4: function B excuting. ディスクアクセスあり. Which function (1-4)...2function B excuting. ディスクアクセスなし、 Which function $(1-4) \dots 1=$ This is function A Which function (1-4)..4 function B excuting. ディスクアクセスなし Which function (1-4)...3This is function C Which function (1-4)...C

A>

リスト 6-3-1 のような構造にしておけば、不要な場合にはディスクアクセスをせずに済みますし、メインプログラムからもいちいち気にせずにオーバーレイ部分のプログラムを一般の関数と同じように呼び出せます。ちょっとした仮想記憶方式になるわけです。ただ、いずれにせよディスクアクセスを伴う場合が存在するわけですから、その関数の実行時間の予測がつきにくく、リアルタイムの制御には向きません。

リスト 6-3-1 の例ではオーバーレイのプログラムを呼び出す場合に引数が設定されておらず、複数の関数を呼び出す複合関数も簡単に作成できるのですが、数多く引数を持つ場合、および引数の数が不定の場合にはこの複合関数の作成方法自体が問題になります。一般に、引数をintなどで受け、実際

の関数呼び出し時に必要なものだけを引数に設定するようにしておけば型は 異なっても値としては同じものになりますので、正しく実行されます。ただ、 引数設定はもともとメモリと実行時間を浪費しますから、オーバーレイする程メ モリに余裕がなくなってきた場合にはなるべく行ないたくないものです。

そのような際にはじかに関数を呼び出す方法をとることもできます。ただしこの場合、基本的にはルートセグメントのプログラムのリンク時にオーバーレイ部分のリンク終了しており、アドレスが決定していなければなりません。ところが、オーバーレイのリンク時には逆にルートセグメントのアドレスがすべて決定していなければならないのですから、どうコンパイルするかが問題になります。

これはなかなか難しいのですが、まず関数名とでためらなアドレスを記述したダミーのシンボルファイルを作成し、リンカーをごまかしてなんとかルートセグメントのCOMファイルを作ってしまいます。その際にシンボルファイルの出力を必ず指定しておき、得られたシンボルファイルを利用してオーバーレイモジュールを作成します。

その際に一s, あるいは一wオプションにより, シンボルリストを出力し, 確定したアドレスを再びダミーのシンボルファイルに書き込み, ルートセグメントをリンクします。オーバレイセグメントのリンク時に出力されるシンボルファイルにはルートセグメントのアドレスも出力されているため, 使うことはできません.

操作例 6-3-2 2回ルートセグメントをリンクする方法

```
exit ();
        while (1)
                printf ("Which function (1-4)..");
                 getline (buf, 3);
                 i f
                         (*buf=='1')
                                          func1 ();
                 else if (*buf=='2')
                                          func2 ():
                else if (*buf=='3')
                                          func3 ():
                 else if (*buf=='4')
                                         func4 ():
        }
A>type func.c
#include <bdscio.h>
main()
                   CLINKでリンクする場合は
                   ダミーで main 関数が必要。
                   (3バイトのロスになる)
func1()
        printf ("function 1 excuting. \n");
func2()
        printf ("This is func 2\fm");
func3()
        printf ("FUNCTION 3, HERE¥n");
func4()
        printf ("this is 4th function\n");
}
A>cc overlay3 -e6000
BD Software C Compiler v1.50a (part I)
  35K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
```

```
あらかじめダミーの
A> type dummy.sym-
                        シンボルファイルを作成しておく.
0000
       FUNC1
0000
        FUNC2
0000
       FUNC3
0000
        FUNC4
                -関数名
                -エントリアドレス
                                    dummy.sym を読み込ませ,シンボルファイル
A>clink overlay3 -y dummy -w -
                                    出力を指定してリンク.
BD Software C Linker
                                    ここで作成された overlay 3. com は実行できな
                                    いので注意。
Linkage complete
  42K left over
                     A>cc func -e6000
BD Software C Compiler v1.50a (part I)
  35K elbowroom
BD Software C Compiler v1.50 (part II)
  32K to spare
A>clink func -v -14000 -y overlay3 -s
                                          ーオーバーレイセグメントのリンク.
BD Software C Linker
                       v1.50
                                   「重複関数あり」というエラーが出力される。
Ignoring duplicate function: FUNC4
                                   多分、FUNC1~FUNC4についてのエラーだ
                                   と思われるが、FUNC4について4回出力され
Ignoring duplicate function: FUNC4
                                   た. CLINKのバグであろう。
Ignoring duplicate function: FUNC4
                                   overlay3. symの中に同じ関数名が存在するた
                                   め、このようなエラーとなるが、func.crl 中
Ignoring duplicate function: FUNC4
                                   の関数が優先されるため問題にならない.
                                                402A FUNC2
1109 CLOSE
                OFB9 EXIT
                                4003 FUNC1
                                OFBC GETLINE
                                                NEBE ISDIGIT
404B FUNC3
                406D FUNC4
DERA ISLOWER
                4000 MAIN
                                OFF8 OPEN
                                                09FA PRINTF
                                                OF50 TOUPPER
110C PUTCHAR
                1060 READ
                                0986 SWAPIN
                                OE16 _USPR
OEED _GV2
                OA1D _SPR
                                                     ~~部のアドレス
                                                    をメモしておく.
Last code address: 4093
Top of memory: DC05
Linkage complete
  46K left over
A>ed dummy.sym
                      dummy.sym を書き換える.
     : *#A
    1: *4T
                FUNC1
    1:
        0000
    2:
        0000
                FUNC2
```

元の内容。

FUNC3

FUNC4

3:

4:

0000

0000

```
1: *I
1:
   4003
            FUNC1
            FUNC2
2:
    402A
                    シンボル出力に従ってアドレスを列挙。
3:
    404B
            FUNC3
4:
    406D
            FUNC4
5:
5: *4K
: *B4T
1:
   4003
            FUNC1
2:
   402A
            FUNC2
3: 404B
            FUNC3
4: 406D
            FUNC4
1: *E
```

```
A>clink overlay3 -y dummy

BD Software C Linker v1.50

Linkage complete

42K left over
```

A>overlay3

```
Which function (1-4)...\frac{1}{2} function 1 excuting. Which function (1-4)...\frac{2}{2} This is func 2 Which function (1-4)...\frac{3}{2} FUNCTION 3, HERE Which function (1-4)...\frac{4}{2} this is 4th function Which function (1-4)...^{C}
```

A>

しかし、この方法は余りにも不便ですし、間違いやすいのでお勧めできませんが、もし用いる場合には、オーバーレイモジュールがロードされているかどうかのチェックはかならず行なってください。

なお、オーバーレイプログラムを作成する場合にはリンカーとして clink 以外にBDS C 付属のL2を用いることができます。考え方は同じですが、オーバーレイセグメントのファイル中に main 関数が無くても良い点、プログラムサイズが若干小さくなる点で有利になります。

あとがき

BDS C, α -C はコンパイル及びリンクが速い C 言語です。本書の執筆には X1turbo + RAM ディスク 2 基という構成で利用しましたが、短いリストはほとんど数秒でコンパイルが終了し、類似の構成で動作させた16ビット用のコンパイラよりも速いのには驚かされました。このため、ちょっとしたプログラム作成には当分手離すことができそうにありません。サブセットであるということ、ソースファイルを一旦メモリにすべて読み込んでから展開するタイプであることがこの高速性を実現しているのでしょう。

C言語の学習という観点から考えるならば、このコンパイル速度はかなり 重要なポイントだと思われます。最初はほとんどがコンパイル&リンクの繰 り返しですから、意欲を失わずに学習効果を上げることができると思います。

なお、本書は私が初心者にC言語を教えるためにまとめたものですが、同時に上級者のためのテクニックも解説した積りです。また、本書でわからないところはマニュアルを必ず参照してください。

さらに高度な部分を知りたい方は拙著「BDS C プログラミング」を併せて 参考にして頂くと良いでしょう。

本書が少しでも読者の方々のお役に立てば幸いです。

御手洗 毅

仕事の関係で、私もC言語を勉強しなければならなくなりました。初めは C言語になじめずとても苦労しましたが、少しずつ理解し、ようやく自分で プログラムを作ることができるようになりました。その際の学習ノートが本 書の原型になっています。 本書は私のテンポにあわせて進みますからおそらく、どなたでも無理なく C言語を学んでいただけるものと思います。私が理解に苦しんだ、関数の中 での引数の扱いかたや、ポインタ、構造体の使い方について詳細に解説され ていると思います。

また、4章以降は内容的に上級者向きになっています。この部分は、何が 書かれているかを読み取ればよいでしょう。この章から長いプログラムがで てきますが、総てを理解する必要はありません。とにかく、そのプログラム が何をするものなのかをつかめば良いと思います。最後に、私に辛抱強く教 えてくれた夫(御手洗 毅)、プログラムを作成するのに時間がかかり、御迷 惑をおかけした工学図書の方々に厚く御礼申し上げます。

御手洗 理英

コンパイルエラーメッセージー覧表

BDS Cおよび α -Cでコンパイル時に発生するエラーのメッセージをABC順にならべてあります。

〈エラーメッセージの見方〉

■インクルードファイルのとき

■ソースファイルのとき

5: Missing semicolon ←ファイルの 5 行め

指定行でエラーが発生したことを示しますが、必らずしもプログラムの誤りはそ の行にあるとは限りません.

Attribute mismatch from previous declaration

異なる構造体などの中で同一のメンバ名を使う場合、その属性は同一でなければならない。

Bad arg to unary operator

単項演算子の使い方の誤り.

Bad argument list

関数の引数中におかしなものがある.

Bad array base

配列として宣言されてないのに、添字を付けている。

Bad case constant

caseの定数がおかしい。

Bad constant

定数がない。

Bad decimal digit

10進定数の中に、数字(0~9)以外の文字がある。

Bad declaration syntax

変数宣言の文法が間違っている。キーワード (char, int 等)を含む文の残りが、宣言文でない時に起こる。

Bad demension value

配列宣言で、サイズを示す添字は、定数または、定数式でなければならない。

Bad expression

式が正しくない.

Bad for syntax

間違った for 文がある. for の () 中には; が2 つなければならない.

Bad function name

関数でないのに関数呼出しを行なおうとした.

Bad left operand in assignment expression

代入式での左項が代入できない定数などになっている.

Bad octal digit

0で始まる 8 進定数に、 $0 \sim 7$ 以外の数字がある時に起こる。

Bad parameter list element

関数定義の引数パラメータの中に, 間違った識別子(変数名)がある.

Bad parameter list syntax

関数定義の引数リスト中にコンマで区切られた変数名以外のものがある。

Bad structure or union member

構造体または、共有体のメンバとして、宣言されてない名前が、ピリオド、または、一>演算子の右側にある。

Bad structure or union specification

構造体または、共有体として、宣言されてない名前が、ピリオド演算子の左 側にある。

Bad subscript

配列の添字がおかしい。

Bad symbols

- y オプションで指定されたシンボルファイルが、シンボルファイルの形式 をしていない。

Bad type in binary operation

2進数演算の中に、おかしなタイプがある。

Bad use of member name

構造体,または,共有体のメンバーとして宣言された識別子は,構造体,または,共有体操作以外では,使うことができない.

Cannot open: (filename)

〈filename〉というファイルが見付からない。

Can't close: (filename)

〈filename〉で示されるファイルが、クローズできない。

Can't create CRL file

出力ドライブのディレクトリ・フル。

Can't find CC2. COM; writing CCI file to disk

CCがCC2を見付けることができず、CCIエクステンションで中間ファイルをディスクに書いた。

Can't have more than one default:

1つのswitch文の中に、1つ以上のdefault:がある。

Close error

ファイルを、クローズしようとした時エラーが起きた。

Compilation aborted by control-C

コンパイル中に、コントロールCがタイプされたらコンパイルは中断する.

Conditional expr bad or beyond implemented subset

#ifなどのプリプロセッサは標準C言語のサブセットである.

CRL Dir overflow : break up source file

1つのソースファイル中に関数の数が多すぎる。ソースファイルを2つに分割したほうがよい。

Curly-braces mismatched somewhere in this function

表示行からはじまる関数に (カーリーブレイス)が多すぎる.

Declaration too complex

このエラーは、関数のレベルが多すぎたり、カッコが多すぎたりした時に起こる。

Dir full

ディスクのディレクトリエリアが、いっぱいなため、ファイルを出力できない。

Disk read error

ディスクからデータを読もうとした時, エラーが起きた.

Duplicate label

ラベルが2重に定義されている。

Encountered EOF unexpectedly (check curly-brace balance)

プログラムが, 途中で終っている. これは, コメントか | が閉じてない時に起こる.

EOF found when expecting # endif

条件コンパル (#if, #ifdef, #ifndef 等) に, #endifがない。

Error on file output disk full?

ファイルにデータを書こうとした時のエラー。ディスクに、空きエリアがない場合が多い。

Error writing: <filename>

〈filename〉で示されるファイルが、書き込みエラーとなった。 ディスク・フルの場合が多い。

Expecting (

普通, while, if, switchキーワードの後の(がない時に起こる。

Expecting:

:がない.

Expecting { in struct or union def 構造体または共有体宣言に { がない.

Expecting { in switch statement switch文の { がない. switch文の式の部分は、複文を含む { } の前になければならない.

Expecting while

do……while文で、whileがない。

Function definition not external

関数定義関数の中にある.

多くの場合、; (セミコロン)を忘れている。

Illegal break or coutinue

break文, または, continue文が間違った位置にある。

Illegal colon

おかしなコロン (:) がある.

Illegal { encountered externally { が、おかしな所にあった時に起こる.

Illegal external statement

関数の外に、おかしな文がある。これは、普通、|が多すぎる時に起こる。

Illegal indirection

ポインターの扱いがおかしい. ポインターではないのに, ポインターとして 使っている変数がある.

Illegal statement

おかしな文がある。

Illegal structure or union

構造体宣言の構造体タグ位置に現われる識別子が、以前に、構造体タグと違うもので宣言されている。

I'm totally confused. Check your control structure!

#include files nested too deep

#includeのファイルが、相互に#includeしている.

Internal error: garbage in file or bug in C

CC2で、このエラーが起きたら、コンパイラのバグと考えられる。BD ソフトウェアに連絡して下さい。

Lvalve needed with $\ ++\$ or $\ --\$ operator

単純変数以外はインクリメント及びデクリメントできない。

Lvalue reguired

代入演算で左項がない.

Mismatched control structure

構造体の制御がおかしい。~~がマッチしていない。

Mismatched parenthesis

カッコがマッチしてない。

Mismatched square brackets

[]がマッチしてない。

Missing from formal parameter list: <name>

変数の宣言が、関数名の後のパラメータリストにない。

Missing function(s): (list-of-names)

(関数名並び)

〈list-of-names〉中に示されるファイルが見つからない。

Missing | in function def.

関数定義中に {がない。プログラム中に、誤った {がない。

Missing legal identifier

必要な識別子が式の中にない。

Missing or misplaced (

(がないか、位置がおかしい。

Missing or misplaced)

)がないか、位置がおかしい。

Missing parameter list

マクロ化された#define中の識別子が、パラメータなしに現われている。 例えば、

#define sqr (A) (A) * (A)

と定義されているのに,

d = sqr

の様に使われている.

Missing semicolon

; がない. しかし, ; がない場合, 他の無意味なエラーをひき起こす場合が 多い.

Need explicit dimension size

配列のサイズが明記されてない。配列宣言でサイズを省くことができるのは、その配列が、関数の引数である時だけである。

Not in a conditional block

これは、 #endifがあるのに、 #if、 #ifdef、 #ifndefがない。

Out of symbol table space; specify more

CCのためのシンボルテーブル領域がオーバーフローした。このエラーが出たら、ソースファイルをいくつかに分けた方がよい。

Parameter mismatch

マクロ化された#define中の識別子の、パラメータの数が合わない。

Redeclaration of : <name>

〈name〉で示される変数が、二重宣言されている。

Sorry; out of memory

ソースファイルが、大きすぎて、メモリーに入らない。このエラーがでたら、 ソースファイルを分割した方がよい。

Sorry, out of memory. Break it up!

ファイルが、大きすぎてメモリ中に入らない。

String overflow; call BDS

#define指定で、たいへん長い識別名を多く持ったために、起こるプリプロセッサ文字列テーブルオーバーフロー。

String too long (or missing quote)

文字列が、長すぎる。(または、゛がない)。

Sub-expression too deeply nested

このエラーは、永久に続く、多重の代入文について発生する.

Syntax error

はっきりしない文法的エラー.

<text>: option error

間違ったオプションが指定された.

The function (foo) is too complex; break it up a bit

関数 foo が長く複雑すぎる。関数を2つに分割しなさい。

Too many cases (200 max per switch)

1つのswitch文の中のcaseが、200を越えている。

Too many dimensions

配列の次元が多すぎる. BDS Cでは、2次元まで.

Too many functions (63 max)

1つのBDS Cソースファイルは、63コの関数までしか、含むことはできない。

Undeclared identifier: <name>

〈name〉で示される名の変数が宣言されてない.

Undefined label used

未定義のラベルが使われている。

Unmatched left parenthesis

(が、合わない。

Unmatched right brace

{がないか、意味のない } があるか。

Warning: Ignoring unknown preprocessor directive

BDS Cでサポートされてないプリプロセッサ指定がある。このプリプロセッサは、無視される。

参考文献

- BDS Cプログラミング: 御手洗 毅 著 (工学図書)
- 2. プログラミング言語 C: B.W.カーニハン/D.M.リッチー 共著 (共立出版) : 石田 晴久 訳
- 3. BDS C Compiler User's Guide: Leor Zolman 著 (ライフボート)
- 4. BDS Cユーザーズマニュアル:リオー・ゾールマン 著 (ライフボート)

索引

《和 文》	【 テ 】 低レベルファイル IO33
[1]	
イニシャライザ62	[/\]
インデント6	倍精度整数62
	バッファードファイル IO33,37,38
	バブルソート23
構造体51	
コマンドライン27	[7]
コメント・・・・・・77	浮動小数点演算77
コンパイルエラー58	プリプロセッサ68,125
[>]	[~]
実数62	ヘッダーファイル84
条件コンパイル・・・・・75	
初期值設定62	【木】
	ポインタ18
[ス]	
スタティック変数62	
1 - 1	
[7]	
チェイン154	

网 格 立	[G]
《欧 文》	getc38
[A]	getcher13
atof ·····79	gets52
atok136	
atokstr135	[H]
	htokana ·····138
[B]	htokstr · · · · · · 138
bdscio. h6	100
break ·····12	[1]
orean 12	iskanji ······140
[C]	
	itof ·····79
cc ····2	r w 1
cc 2 ·····2	[K]
ck157	kanatoh ······139
clink ······3	kgetline ······134
close34	ktoa ·····136
cos90	ktoastr·····136
CPMEOF41	ktoc125
creat34	ktohstr139
	ktype139
(E)	
ED2,5	[L]
EOF41	[∟] left ·····47
ERROR41	long62
execv155	
exit36	[M]
	main6
[F]	mid ······49
FALSE41	
FCH ·····130	[N]
fclose38	[N] NULL41
fcopy89	
fcreat ·····38	[0]
fgets38,39	[O]
FILE39	·
float ·····62,77	[P]
fopen38	printf7,30
for10	putc38
fpadd79	putchar ·····13
fprintf38,39	
fputs38	[Q]
fscanf ·····38	qsort27

[R]	swapin159
read34	
right48	[T]
romanstr137	tan93
	TRUE41
[S]	
scanf12,52	[U]
seek34	unlink103
sin ·····86	
strcat44	[W]
strcmp45	while42
strcpy······44	write34
strlen46	
struct53	[X]
SUBMIT96	XSUB96
≪その他≫	# define
/106	# endif68
++21	# if68.75
¥" ·····31	# ifdef ······68,75
¥n	# ifndef75
¥t12	# include
\$\$\$. SUB99	# undef68.73
%······11	&12
%d ······31	@·····96
704	90

-BDS C/α-C関連ソフトウェアについて-

(有)アーマットではBDS C/α -C関連ソフトウェアとして「BDS C ユーティリティパッケージ」を発売しています。このパッケージの中には、Z80 エーモニック用のアセンブラプリプロセッサ・Z (Z (Z (Z) などが収められており、Z (Z) のトレンア・Z (Z) のトレンア・Z (Z) のトルスを有効な内容になっています。これらのソフトウェアについての評細は、工学図書発行の「BDS Z) のアログラミング」をご覧ください。なお、本書で紹介した「日本語処理パッケージ」、「拡張サブミットコマンド」も含まれておりますので、本書のディスクサービスとしてもご利用ください。

現金書留か郵便振替で住所,氏名,年齢,勤務先,電話番号,使用システム,ディスクタイプを明記の上,下記宛にお申し込み下さい.販売は通信販売のみです.

ディスク内容

fplong.h float, long関数のヘッダーファイル

zcasm.c casmのZ80版ソースプログラム

zcasm.com casmのZ80版実行ファイル

mrasm.com Z80ニーモニックのアプソリュートアセンブラ(VI.3)

dacrl.com CRLファイルをZ80アセンブラのソースファイルに

変換するユーティリティ

nihongo.c 日本語処理ユーティリティ

ktoc.c 日本語変換ユーティリティ

exsub.c (@.com) 拡張サブミットコマンド (ソースファイルと実行ファイル) **exsub2.c** (/.com) 拡張サブミットコマンド (ソースファイルと実行ファイル)

その他、「BDS Cプログラミング」のディスクサービスで扱っていたものはすべて含まれます。

価格

10,000円 (送料を含む)

対象システム

PC-8001, 8801シリーズ(5インチ2D) X1, X1 turboシリーズ(5インチ2D), CP/M標準(8インチ片面単密), その他のシステムの方は, あらかじめお問い合わせください.

申し込み先

〒227 横浜市緑区荏田町473-5

(前)アーマット TEL. 045-911-7427 郵便振替 横浜5-30518

■著者略歴

御手洗 毅(みたらいつよし)

昭和30年9月24日生れ

昭和53年 慶応義塾大学工学部計測工学科卒

昭和53年 カシオ計算機㈱入社後電子楽器開発に従事。

昭和58年 (株)フジテレビジョン入社 (CG 担当)

昭和61年 (前アーマット設立

著書 「スーパーインポーズ CP/M」(工学図書) 「BDS Cプログラミング」 (工学図書)

御手洗 理英(みたらいりえ)

昭和35年3月18日生れ

昭和56年 駿台電算専門学校卒

昭和56年 (㈱フジミック入社後ソフトウェア開発に従事

昭和61年 (旬アーマット

現在 東京工業専門学校講師

著者の承 認により 検印省略

学習 BDS C·α-C

昭和62年4月6日 初 版 昭和62年6月30日 第 2 版

著者衛手洗穀

理 英

発行者 小 林 泰 明

発 行 所 工学図書株式会社 (営業所)東京都千代田区九段北1-14-15 〒102 電話 東京(262)3772番

振 替 東 京 7-13465 番

印刷所株式会社サンヨー

C 御手洗毅, 御手洗理英 1987

定価 1,900円

ISBN4-7692-0164-8 C3058

BDS Cプログラミング

御手洗殺 著 A 5・280頁 定価2600円

<内容> BDS Cコンパイラ入門/コンパイラの使い方/BDS Cライブラリ/プログラムパッケージ/BDS Cとアセンブラのリンク/BDS C技術情報/ROM化の方法/あとがき/付録/参考文献/索引

スーパーインポーズCP/M-CP/Mを用いたマシン語プログラム作成入門-

御手洗毅 著 A 5・240頁 定価2400円

<内容> CP/Mのディスク整理/アセンブラプログラミング実習/CP/Mの特徴と問題点/アセンブラによるより高度なプログラム開発/DDTによるデバック/BASICとのリンク/CP/Mでのプログラミング/MR-FORTH/CP/Mの改造/付録

X1+turboマシン語読本

清水保弘 著 A 5·268頁 定価2400円

<内容> マキン語を始めよう/Z80 CPUとハンド・アセンブル/データの転送/マシン語での計算/プログラムの流れをかえる/入出力と画面制御/桁ずらし、回転、ピット操作/交換命令/CPU制御命令/BASICとマシン語の共存

マイクロコンピュータ基礎用語辞典

渡辺富夫 著 B 6 · 190頁 定価1200円

本書は、マイクロコンピュータを習得するにあたり、必要と思われる用語約360語を厳選し、他の用語との関連も十分に考慮して、出来る限り平易に解説している。特に重要な用語に対しては重点的に詳述している。また、各用語は、その用語のレベルに合わせて解説され、マニュアル等にも明白に述べられていない点も言及されている。したがって、マイクロコンピュータを初めて学はれる方にも、既に相当のレベルに達している方にも、本書は充分に役立つように配慮されている。

学習 α · COBOL

坪根 斉·菅原光政 共著 A 5·170頁 定価1800円

<内容>COBOLプログラミングとは/COBOLプログラミングの書き方/簡単な計算するプログラムの書き方/簡単な報告書作成プログラムの書き方/実行の流れを変えるプログラムの書き方/くり返し処理を行うプログラムの書き方/表を用いたプログラムの書き方/いろいろなプログラムの書き方/簡単なグラフ作成/総合問題/付録

実習 α-FORTRAN

小坂貴美男 著 A 5·176頁 定価1800円

<内容> ■第1部: α -FORTRANとの出会い/CP/M入門/ α -FORTRANの準備/ α -FORTRANの使い方 ■第2部:FORTRANプログラムの基礎知識/データの出力/データの入力/FORTRANの計算式/判断と分岐/繰り返し処理/配列の使い方/ディスクへの入出力/副プログラム ■付録: α -FORTRAN便利帳/練習問題の解答例 ■索引

α -PASCALプログラミング

杉原敏夫・菅原光政 共著 A 5・184頁 定価1800円

〈内容〉 まずプログラムを作って動かしてみよう/PASCALプログラムの書き方/データの型/文とプログラミング/手続きと関数/データ構造とその型/プログラミング例題/ α -PASCALの特徴/付録

〒102 東京都千代田区 九段北1-14-15 **工学図書株式会社** 電話(03)262-3772 振替 東京7-13465









